UNITED STATES PATENT APPLICATION

OF

CURTIS GENEROUS,

RICHARD DUNBAR,

JERRY RUSNOCK,

MATTHEW WHALEN

AND

CHRISTOPHER SHENTON

FOR

MULTI-CHANNEL MESSAGING SYSTEM AND METHOD

[0001] This application claims priority to U.S. Provisional Application No. 60/224,507, filed on August 14, 2000, and to U.S. Patent Application No. _____, entitled MULTI-CHANNEL MESSAGING SYSTEM AND METHOD, filed on August 14, 2001, which are both incorporated by reference herein.

[0002] This application includes material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent disclosure, as it appears in the Patent and Trademark Office files or records, but otherwise reserves all copyright rights whatsoever.

BACKGROUND OF THE INVENTION

Filed of the Invention

[0003] This invention relates to delivery of electronic information to its intended recipient over a network, and more particularly, to delivery of information, such as a message, over multiple communication channels in a manner that increases the likelihood of receipt by the recipient.

Discussion of the related art

[0004] Although a number of methods currently exist to deliver information, such as email, to a recipient, these methods suffer from a number of drawbacks. One of these drawbacks is a general difficulty in the sender knowing whether and when the intended recipient received the message, particularly when the information is time sensitive, and the recipient is capable of receiving the message on multiple communications channels, such as email, voice message, Instant Messenger, pager, etc.

[0005] Accordingly, there is a need for a system and method that substantially increases the likelihood of receipt of the message in the shortest possible time.

SUMMARY OF THE INVENTION

[0006] In a preferred embodiment, the invention provides a high-performance message distribution engine for delivering large volumes of customized messages via multiple delivery channels, tracking message queuing and delivery status, tracking message access by recipients, performing automatic alternate delivery of messages upon encountering errors, performing automatic alternate delivery of messages when recipient fails to read the message within a pre-configured amount of time, and providing immediate and summary reports to clients and administrators of the system.

[0007] Alternate "Delivery Agents" are provided for delivering messages to a recipient via alternate paths, such as e-mail, pager, voice, FAX, or instant message. A "Message Rollover" or "Message Roaming" function is provided for sending a message to a recipient using alternate e-mail addresses or message delivery agents if the primary delivery agent fails to deliver the message or the message is not acknowledged within a pre-determined amount of time.

[0008] The system of the invention encompasses a rules-based delivery engine, which provides for a per-recipient profile database, which allows for an automated selection of the optimal delivery channel, based on such factors as priority of message, time of day, type of information, past delivery history, type of client software used, etc... The system of invention includes a system architecture which is highly scalable, fault tolerant, and supports geographical diversity and a fully distributed architecture, capable of high volumes of traffic with respect to each of the above functions. The system of the invention can be used to distribute large volumes of messages, newsletters, alerts, electronic documents, and other multimedia information. Interfaces are provided for allowing third-party clients to use the system to send large volumes of

messages and receive reports on the delivery status of their customizable messages. This and other auditing, tracking and reporting functionality allows the system in its preferred embodiment to be operated as a service bureau.

BRIEF DESCRIPTION OF THE ATTACHED DRAWINGS

[0009] The foregoing and other objects, features, and advantages of the invention will be apparent from the following more particular description of preferred embodiments as illustrated in the accompanying drawings, in which reference characters refer to the same parts throughout the various views. The drawings are not necessarily to scale, emphasis instead being placed upon illustrating principles of the invention.

[00010] FIG. 1 shows a diagram illustrating the message assembly and delivery processes of the invention in accordance with a preferred embodiment.

[00011] FIG. 2 shows a diagram illustrating an overview of the architecture of the system of the invention in accordance with a preferred embodiment.

[00012] FIG. 3 shows a diagram illustrating the reporting subsystem and related subsystems in accordance with a preferred embodiment of the invention.

[00013] FIG. 4 shows a diagram illustrating the reporting subsystem and its data store according to a preferred embodiment of the invention.

[00014] FIG. 5 shows a diagram illustrating the assembly engine and its related subsystems in accordance with a preferred embodiment of the invention.

[00015] FIG. 6 shows a diagram illustrating the tracking subsystem of the invention according to a preferred embodiment.

[00016] FIG. 7 shows a diagram illustrating the instant messenger subsystem of the invention according to a preferred embodiment.

[00017] FIG. 8 shows a diagram illustrating process distribution in the system of the invention according to a preferred embodiment.

[00018] FIG. 9 shows a diagram illustrating service management in the system of the invention according to a preferred embodiment.

[00019] FIG. 10 shows a diagram illustrating the database architecture of the system of the invention according to a preferred embodiment.

[00020] FIG. 11 shows a diagram illustrating the web server architecture of the system of the invention according to a preferred embodiment.

[00021] FIG. 12 shows a diagram illustrating the application server architecture of the system of the invention according to a preferred embodiment.

[00022] FIG. 13 shows a diagram illustrating the delivery agent architecture of the system of the invention according to a preferred embodiment.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[00023] The system of the invention in its preferred embodiment provides operations which encompass a complete lifecycle of electronic communications, including 'mailing' initiation, pre-processing and message assembly, scheduling, delivery, tracking and reporting. Certain broad features of the invention will be discussed firstly below.

[00024] **Message Priority**

[00025] The system of the invention provides for individual message priorities to be set, and for that information to be used to determine the delivery sequence of these messages. Multiple priority levels for messages may be supported in the system of the invention. These priorities may be, e.g., Bulk, Normal, Alert, Blast, and Priority -[1-5]. Higher priority messages will always take precedence over lower priority messages. The system of the invention uses the

priority information of the message to determine the order in which a message should be processed relative to other jobs being performed Message Priority for delivery is discussed in detail in the E-mail Delivery Agent (EDA) Subsystem section below. The user/customer provides a priority for all messages in a submission and that priority will be assigned to each message.

[00026] **Rollover or Message Roaming**

[00027] The "Rollover" or "Message Roaming" feature of the invention provides the ability to send a message to a recipient using multiple alternate delivery channels or message delivery agents should the primary delivery agent fail to deliver the message or not be acknowledged within a specific time frame. Rollover will occur for a message if specified in client input for a recipient and if the message delivery fails for the primary or more preferred delivery agent. Message delivery failure may be determined, e.g., based upon feedback from remote message transfer agents (MTAs), discussed in further detail below. Multiple levels of rollover are preferably supported.

[00028] The message roaming feature of the invention provides for a much greater delivery completion success by providing the ability for multiple delivery channels to be tried, either in succession (Alert) or in parallel (Blast), until a success condition has been met. The trigger mechanism for the rollover feature to be activated include such conditions as:

[00029] • Failure reported by a delivery agent (DA) when attempting to deliver a message. Failures are classified by either a hard-error (permanent) or soft-error (temporary). Different actions can be taken when encountering either hard or soft errors.

[00030] • Inability of a delivery agent to contact the remote device after a pre-determined amount of time (time based rollover or TBR)

[00031] • The ability to trigger an alternate delivery channel based on the user's profile and/or based on policy / rules (e.g. send information to a pager if a critical message is sent during non-business hours).

[00032] **Unified Message Tracking**

[00033] The system of the invention provides for the ability to gather real time statistics on the actions of delivery, reading, bouncing, forwarding, in-transit, queued, and replies. The status of each message is tracked so that clients may query for the status of messages and for summary reporting of submissions. The system should be able to trace message assembly, queuing, delivery and failures, to remote destinations, independently of the communication channel used.

[00034] The date and time the message was read should also be tracked. These data are recorded each time a message is read.

[00035] The recipient's software provides functionality for reading the message, time message was read, message size, network location, connectivity speed and any other attributes that can be gleaned about the user. Data extracted from a recipients' browser are collected by a Web Agent (WAG). The WAG sub-system is described in further detail below; WAG Subsystems requirements are set forth below and identify which information is to be tracked.

[00036] An Out of Band Subsystem receives and initiates tracking of bounced e-mail and autoreply messages.

[00037] Both Variable Envelope Return Path (VERP) and Delivery Status Notification (DSN) are preferably supported. Known ways to track messages should be supported. Their use may be configurable by a Systems Administrator where necessary. DSN "success" messages may be used but may not necessarily be "on" by default.

[00038] The use of delivery channel-specific VERPs are the preferred principle method used to track and identify errors relating to a message. All information necessary in processing and tracking requests must be made available by knowing the VERP.

[00039] **Message Transfer Agents Probes / Health Probes**

[00040] Independent software entities may be provided which periodically connect to, and sense the operational status of, remote message transfer agents (MTAs) throughout the top one thousand domains in use by the system. Connection times, MX server preferences, etc., can all be recorded and analyzed to provide a set of 'best bet' MTA server addresses for each domain at any given time prior to the time when they are needed by the DAs. This set of MTA server addresses can be referenced instead of the more time-consuming DNS for 95+% of all outbound mail. Additional health-probes may be used to determine the operational status of other remote servers (e.g. AOL AIM servers) and used by the appropriate DA's.

[00041] **Message Expiration Time**

[00042] All messages within a single submission preferably have one and the same associated expiration time. The default expiration time for the system may be set at, e.g., three days from the time of scheduled delivery. Expiration date and time are specified in Universal Time Coordinated (UTC). Values for expiration time should be able to be configured at the system level, at the client level and at the submission level.

[00043] Messages that exceed the expiration time set for them are deleted from the message queue. Messages are not assembled if the anticipated assembly time exceeds the Expiration Time. Tracking information is submitted to the Delivery Scheduler for entry into the Tracking Database tables for each message deleted from the message queue or not assembled due to exceeding expiration time.

[00044] **Custom Messaging**

[00045] Clients/users of the system are provided with the ability to customize messages sent to their recipients on a per-user basis for content in the message by using variable substitution. Global defaults may be set for all messages in a submission. Per- recipient values will always supercede global values. Clients are able to customize message per Delivery Type on a per-user basis. There is preferably one, non-recursive level of substitution done per message. The customization data is provided in the client's submission. The following data elements may need to be accommodated in the submission:

[00046] • Recipient's zip code

[00047] • Recipient's city

[00048] • Recipient's state

[00049] • Recipient's area code

[00050] • Recipient's name

[00051] • Recipient's delivery time

[00052] **Secure Encrypted Mail**

[00053] The system of the invention includes functionality that provides for various security features to be associated with a message.

[00054] Messages sent by a user of the system may be encrypted, either by the user or by the system itself, and mechanisms for decrypting messages at the recipient end once the message is received.

[00055] Messages sent by the system may include 'conditional' events that determine whether or not a message is able to be read by the recipient. The possible conditions may include, in part, such variables as:

[00056] ▪ Was the message received by a authenticated recipient, using the PKI infrastructure and digital certificates such as those commercially available by organizations like Verisign. The use of S/MIME is envisioned as the preferred mechanism for this purpose.

[00057] ▪ Is the message attempted to be read by an approved IP address?

[00058] ▪ Is the message attempted to be read within a preset time frame? Within specific time windows (e.g. business hours only)?

[00059] ▪ Is the message attempted to be read originating from a pre-approved software module (e.g. allow message to be read by a Eudora mail client but not by an AOL mail client).

[00060] ▪ Comparing the source of the mail read request to previous successful requests and deny if source if different.

[00061] **Shreddable Mail**

[00062] The system of the invention may include functionality which allows a sender to associate rules with a message to determine if it can be opened and read by the recipient. In this respect, the system allows the sender to determine that messages which do not meet the rules (e.g., they are not delivered or read before a certain date) are automatically deleted. Some of the rules that may be used include:

[00063] ▪ Time Lapse: message must be read within a certain time frame or it becomes unavailable (ala 'shredded' metaphor)

[00064] ▪ Delete after read once (i.e. mission impossible metaphor: 'this message will self destroy in 5 seconds')

[00065] ▪ Who reads it: require that the message be read from a specific device (e.g. IP address) . Password access can be enabled.

[00066] In one embodiment, messages containing only a URL are delivered to a recipient; the content is available only via click through to the Web Delivery Agents (discussed in further detail below) and may, therefore, be deleted under certain conditions or at certain times, e.g., when the message is read once or after a certain time has elapsed.

[00067] **Recipient Data and Client List Management**

[00068] Clients/users of the system are responsible for recipient list management and all messages submitted to recipients. The client provides subscribe/unsubscribe information in each message. That information preferably includes either a URL or an e-mail address that can be used by a recipient to subscribe/unsubscribe. A client is able to send messages to its full list of recipients, but is preferably excluded from sending messages to a subset of their recipient list. A client who wishes to send messages to subsets of their recipient list is thereby forced to separately register each submission.

[00069] Recipient data is persistent across all clients and submissions. A client may given the ability, through a directive in a submission, to request that the recipient list not be persistent. A client's recipient list data will be persistent across submissions. A client's recipient list data may point to global recipient data. There will be a "blacklist" of domains and recipients which are not to receive messages.

[00070] **Message Delivery by Recipient's Geographic Location**

[00071] Messages for recipients are preferably deliverable based on the client's input of geographical information included within the submission input. The Delivery Scheduler provides functionality to segregate messages based on the attributes listed below and sent to a specific Assembly Engine for assembly and delivery based on local time of the recipient.

[00072] Segregation attributes include, but are not limited to, the following:

[00073] • ZIP code

[00074] • City

[00075] • State

[00076] • Country

[00077] • Phone number Area Code

[00078] • Time zone data (e.g. EDT, PST)

[00079] • Latitude / Longitude data

[00080] An example of this use would be a client who wants all messages delivered by 8:00 am local time for the recipient. Delivery of the messages could be split up as to deliver Easternmost time zone messages first and then follow with following time zones. Messages for other time zones might be transferred to a remote location for assembly and delivery.

[00081] **Guaranteed Mail**

[00082] In accordance with one feature of the invention, functionality is provided whereby operators of the system can guarantee that a message will be delivered to its recipient, except where the recipient's id no longer exists or is unreachable, etc. In this respect, the system may include functionality for embedding special data tags in electronic messages that trigger specific, detectable events to occur on the system's servers. These data triggers are used to record all activity associated with the 'history' of a message, including whether or not a message has been received and acted upon by a recipient. This, combined with the Out Of Band (OOB) error handler, provides the ability to 'guarantee' to a sender that a message was successfully received and read by a recipient.

[00083] **OVERALL SYSTEM OPERATION**

[00084] FIG. 1 shows the message assembly and delivery processes of the invention in accordance with a preferred embodiment. At "Input", submission data is accepted in XML format. The XML Parser then parses the input data, stores the parsed data in the Submission Database, stores the unparsed XML input file in the Archive Database, and informs a Submission Router of the new submission. The XML Parser creates the submission ID for the new submission.

[00085] Submission Routers are provided for keeping track of which Delivery Scheduler is handling a particular submission. The Submission Router is responsible for deciding which Delivery Scheduler (discussed in detail further below) should handle a particular submission. This decision is stored in the database for future reference by other Submission Routers. The Submission Router should communicate with that Delivery Scheduler and inform it that it is responsible for that submission.

[00086] If a given Delivery Scheduler should become unavailable, the Submission Router should find another Delivery Scheduler to take over all of the submissions currently being handled by the failed Delivery Scheduler. The Submission Router should notify the new Delivery Scheduler that it is now tasked with handling the submission.

[00087] In order to meet the above obligation, it may be necessary for the Submission Routers to monitor the health of the various Delivery Schedulers. They should not only monitor whether they are up or down, but how busy they are.

[00088] If a process asks the Submission Router which Delivery Scheduler is responsible for a particular submission, the Submission Router checks it own cache to determine the correct answer. If the answer is not in the cache, the Submission Router gets the answer from the database.

[00089] The Delivery Schedulers are provided for keeping track of the status of each message for a particular submission. A Delivery Scheduler initiates assembly and delivery as needed. The Delivery Scheduler keeps track of the assembly status for each message of a particular submission that is scheduled for future delivery. The Delivery Scheduler keeps track of when submissions should run. A given submission is handled entirely by a particular Delivery Scheduler.

[00090] When the Delivery Scheduler is told that it is responsible for a submission, it spawns a new thread, retrieves all of the submission data from the database and builds it own internal data structures.

[00091] If any process in the system comes to the conclusion that a particular recipient should have their message sent via their second or third delivery option, that information is directly passed to the appropriate Delivery Scheduler. That process will ask the Submission Routers which Delivery Scheduler is handling that submission.

[00092] If any process determines that a particular recipient has received their message and no more messages should be sent, that information is directly passed to the appropriate Delivery Scheduler.

[00093] When the Delivery Scheduler talks to a particular Assembly Engine, it tells the Assembly Engine whether the data should be handed to a Delivery Agent immediately, or whether it should be stored in the Assembled Messages Data Store (for pre-assembled submissions).

[00094] The Delivery Scheduler submits a list of recipients (typically 1,000 at a time) that need to be assembled to any given Assembly Engine. The Assembly Engine then retrieves

message templates, message style sheets, and all of the recipient information directly from the submission database.

[00095] Submissions that should not be delivered immediately are put into a timer queue. The messages are all assembled. The data structure holding the submission linked list is deleted. When the time comes to deliver the submission, the database is queried and the submission linked list is rebuilt. The delivery is then started.

[00096] With continuing reference to FIG. 1, an Assembly Engine takes the message template, the message style sheet and recipient information and combines them to create the message which is to be delivered. Any process that communicates with the Assembly Engine tells the Assembly Engine to either store the assembled message in the Assembled Messages Data Store or to contact a particular type of Delivery Agent.

[00097] Any process that communicates with the Assembly Engine tells the Assembly Engine whether that particular message has been pre-assembled. If the message has been pre-assembled, the Assembly Engine merely reads the message from the Assembled Message Data Store. If the message has not been pre-assembled, then the Assembly Engine assembles the message.

[00098] Any process that communicates with the Assembly Engine tells the Assembly Engine the range of recipient IDs which are to be assembled.

[00099] An Assembled Messages Data Store is provided for storing pre-assembled messages. A hashing function should be used to determine where to store the assembled message. This should be a function of the recipient ID and the delivery option.

[000100] Delivery Agents are provided for receiving assembled message and delivery information and delivering the message. The messages are dropped into queues and delivered as

soon as possible. If a delivery attempt fails fatally, that information is communicated to the appropriate Delivery Scheduler.

[000101] Web Delivery Agents are provided for delivering an assembled web message to a client. The Web Delivery Agent should directly notify the appropriate Delivery Scheduler that the recipient has asked for the web message.

[000102] A web tracking agent is provided for reporting the retrieval of tracking images embedded in messages. The Web Tracking Agent should directly notify the appropriate Delivery Scheduler that the recipient has asked for the web message.

[000103] An Out-Of-Bounds (OOB) handler is provided for handling out-of-bounds messages. When a message bounces, 3rd party ACK, ASCII email ACK, or a Delivery Status Notification (DSN) arrives, it is received by the OOB Handler. The OOB Handler notifies the Delivery Scheduler of the bounce so that the Delivery Scheduler can immediately initiate the assembly/delivery of the second or third delivery option.

[000104] **SUBSYSTEMS**

[000105] Diagrams of the Subsystems of the invention in a preferred embodiment appear in FIGS. 2 through 7.

[000106] The system's engine in its preferred embodiment is comprised of the following subsystems:

[000107] • Data Upload

[000108] • Parse Validate and Load

[000109] • Submission Router

[000110] • Delivery Scheduler

[000111] • Assembly Engine

[000112] • Administrative Interface

[000113] • Reporting (Internal & External)

[000114] • Wag

[000115] • EDA

[000116] • IMDA

[000117] • DAStats

[000118] • MTA Probe (MTAP)

[000119] • Logging

[000120] • Tracking

[000121] Each of these subsystems will be described in detail further below.

[000122] **Subsystem State Information**

[000123] Every subsystem writes its state information to a database or to a file on an NFS-mounted file system. If the data are on an NFS, then MAILDIR format will be used. The data will be used by the Administrative Interface for displaying subsystem status. At a minimum, the following state information is required:

[000124] • Subsystem name

[000125] • Process ID of the parent process

[000126] • State (up/down/starting/stopping)

[000127] • Last update time

[000128] • General metrics that will indicate load

[000129] The update interval is specified in a Configuration Database, discussed below. All state changes will require an update. The lack of any update within a period that is twice the

length of the specified update interval will be interpreted as an indication that the particular subsystem is down.

[000130] All configuration information for all subsystems is preferably stored in a Configuration Database. All subsystems obtain their configuration information from this Configuration Database.

[000131] All daemons preferably follow general Unix conventions for responding to signals. These conventions include the following:

[000132] • HUP (1): Read the configuration file and reinitialize

[000133] • TERM (15): Shutdown

[000134] • USR1 (16): Increment debug by one level

[000135] • USR2 (17): Halt debug

[000136] • STOP: Suspend

[000137] • CONT: Continue

[000138] All signals should be handled consistently across all subsystems. A consistent and unique signal should be used to dump state information to a flat file.

[000139] Each subsystem writes its state information to a database or to a file on an NFS-mounted file system. If the data are on an NFS, then MAILDIR format may be used. The data is used by the Administrative Interface for displaying subsystem status. At a minimum, the following state information will be required:

[000140] • Subsystem name

[000141] • Process ID of the parent process

[000142] • State (up/down/starting/stopping)

[000143] • Last update time

[000144] • General metrics that will indicate load

[000145] • Uptime

[000146] The update interval is specified in the Configuration Database. All state changes generally require an update.

[000147] The system of the invention preferably includes functionality for processing and responding to automated response messages from recipients.

[000148] All blocking calls (reads/writes) should be wrapped with a SIGALARM to prevent deadlocks.

[000149] With continuing reference to FIGS. 2 through 7, each subsystem will be discussed in detail below.

[000150] **Logging Subsystem**

[000151] A logging subsystem is provided for logging system events and errors to log files. The logging Subsystem should rotate logs and remove old logs based on the configuration information as specified in the initialization function. Log rotation preferably occurs as part of re-initialization after the reception of a HUP signal. The logging Subsystem should be re-entrant, thread-safe and as non-blocking as possible.

[000152] If writing a log message or opening a log file fails for any system administrator recoverable problem, then block all logging calls from returning control to the application and send the error to standard error and the SNMP monitoring station with an Emergency message level.

[000153] A Logging library is preferably provided and can be linked with any application or subsystem of the system of the invention. The logging library should contain an initialization function which accepts information on the service name (what the calling service will be called

in the logs) to write in the log file. The logging library contains functions to write messages to logs. Each function accepts a severity level, a debug level to determine if the log message should be written or ignored and a variable number of parameters with substitutions to define the message. The logging library preferably only logs full lines. It should be re-entrant, thread-safe and as non-blocking as possible. If writing a log message or opening a log file fails for any system administrator recoverable problem, then all logging calls are blocked from returning control to the application and the error is sent to standard error and the SNMP monitoring station with an Emergency message level.

[000154] The severity levels that the logging Subsystem should understand are the same as syslog - DEBUG, INFO, NOTICE, WARNING, ERROR, CRITICAL, ALERT, and EMERGENCY.

[000155] The format of a log file line is preferably as follows:

[000156] YYYY/MM/DD HH:MM:SS hostname service[pid] severity: message filename:linenumber

[000157] The logging library preferably contains functions which accept a service name, message severity level, debug level, and log message. Based on the inputs, the log library will determine if the message should be written to a log file. If the message is written to a file, the library will construct the log message line and send an SNMP trap if appropriate for the message based on the severity. The log library will send messages to a message queue for being written to disk. A success or failure message will be sent to the calling application.

[000158] **LOGGING DAEMON**

[000159] A Logging Daemon is provided for consolidating messages for all Ranger (eFoton) processes running on the system(s). The logging Subsystem preferably contains an

initialization function which accepts information on directory in which to store logs, log filename, service name to write in the log file, retention length before removal of old log files, rotation frequency in minutes, and size at which to rotate files. Default values are: log rotation frequency are 60 minutes, size at which to rotate files is 10MB, directory is "/var/log", filename is the same as service name. Service name is not optional.

[000160] The logging Subsystem preferably sends an SNMP trap to the monitoring station based on a configuration parameter. If not specified in the configuration database, messages of severity level "Warning" or higher will result in SNMP traps being delivered to the SNMP monitoring station. [i.e. TrapDebug = FALSE, TrapError = TRUE]. After being rotated, logs will be compressed using *a compression utility.*

[000161] The log daemon receives input from the configuration database and the log message queue. Optionally, the log daemon may receive messages via a TCP connection from other log daemons. The log daemon writes the log messages to the appropriate file or sends the messages to another log daemon via TCP for writing.

[000162] **Tracking Subsystem**

[000163] The invention includes a Tracking Subsystem for collecting and storing tracking data from the Ranger subsystems to allow tracking of messages from submission, through processing, to delivery to, and reading by, the recipient. The tracking subsystem preferably comprises the following elements:

[000164] • Tracking Library

[000165] • Tracking Daemon

[000166] • Tracking Data Store

[000167] These elements are illustrated in FIG. 6, and are described below.

[000168] The tracking library is a library that can be linked with any application or subsystem of the invention. The tracking library is be re-entrant, thread-safe and as non-blocking as possible. If writing to the tracking data store fails for any reason, control is returned to the application, the error is recorded in the Subsystem State Table and the Logging Subsystem sends an SNMP trap at the EMERGENCY level to the Network Monitoring System.

[000169] The status of each message is tracked so that clients may query for the status of messages and for summary reporting of submissions. The system should be able to trace message assembly, queuing attempted delivery, delivery, message expiration and rollover. The date and time the message was read will be tracked if at all possible; advertisements, messages, and submission progress will also be tracked.

[000170] The recipient's software for reading the message, time message was read, message size, network location, connectivity speed and any other attributes that can be gleaned about the recipient. Data extracted from a recipients' browser should be collected by the Web Agent (WAG). The WAG sub-system is described in detail below.

[000171] The Out of Band Sub-system will update tracking of bounced messages and messages generated by auto-responders.

[000172] The following data will be stored in the tracking database. Data will be submitted to the Submission Router by the Delivery Agent.

[000173] • MessageID

[000174] • Success / hard failure

[000175] • Reason for failure

[000176] • MTA speed in bytes per second

[000177] • Date & time recorded in UTC. The date should include a 4-digit year. Time will be recorded to the millisecond, and if possible, to the microsecond.

[000178] • RecipientID

[000179] • SubmissionID

[000180] The Tracking Subsystem updates the Message Table with the following

[000181] on behalf of all DAs upon receipt of a delivery success / hard fail:

[000182] • Status (delivered / failed / trying alternate DA)

[000183] • UpdateDt (record update date-timestamp

[000184] • DaType DA which delivered or last DA tried

[000185] The tracking library contains functions accepting tracking data to be sent to the tracking daemon. The data includes the following data elements (not inclusive) from the subsystems for storage in the tracking data store:

[000186] ▪ trackingID- tracking module generated

[000187] ▪ trackingDate

[000188] ▪ trackingTime

[000189] ▪ clientID

[000190] ▪ submissionID

[000191] ▪ messageID

[000192] ▪ recipientID

[000193] ▪ module- submission router, parse/validate, assembly, E-mailDA, IMDA, WAG, OOB

[000194] ▪ moduleStatus- status of module processing, possible valid values (not inclusive):

[000195] - MTA accepted

[000196] - MTA deferred

[000197] - MTA reject

[000198] - Ranger queued

[000199] - Deferred queue

[000200] - Roll over attempted

[000201] - IM expired

[000202] - Ad expired

[000203] - Delivery success

[000204] - Delivery failure

[000205] - Message read

[000206] - Received

[000207] - Received format check-ok

[000208] - Assembly scheduled

[000209] - Assembly started

[000210] - Assembly complete

[000211] - Delivery started

[000212] - Delivery complete

[000213] - Message delivered

[000214] - Message assembled

[000215] - Message delivery failed - retry

[000216] - Message delivery failed - rollover

[000217] - Message delivery failed - hard

[000218] - Message bounced

[000219] - Message deferred

[000220] - Message read

[000221] - Administratively deleted

[000222] - Error code

[000223] - MessageReadTime

[000224] - MessageReadDate

[000225] - MessageDeliveryCost

[000226] - DeliveryClass (E-mail, IM, Pager, WAG)

[000227] - E-mailAddress

[000228] - Domain

[000229] - Mua

[000230] - IspMTA

[000231] - Browser

[000232] - RecipientConnSpeed

[000233] - IspConnSpeed

[000234] - ImURL

[000235] - ImID

[000236] - SuccessDeliveryDate

[000237] - SuccessDeliveryTime

[000238] - Ad clicked

[000239] - moduleStatusValue

[000240] As set forth above, the tracking library contains functions that send messages to the tracking daemon. Success or failure indication is returned to the calling application. Data sent to the tracking daemon includes the Input data elements described above.

[000241] A tracking daemon is provided which accepts tracking messages from all Ranger processes running on the system(s). The Tracking Subsystem updates the Message Table with the following data on behalf of all Delivery Agents upon receipt of a delivery success / hard fail:

[000242] • Status (delivered / failed / trying alternate DA)

[000243] • UpdateDt (record update date-timestamp

[000244] • DaType DA which delivered or last DA tried

[000245] The tracking daemon receives configuration information from the configuration database. The tracking daemon is also preferably able to receive messages from a TCP connection from remote systems of the invention. The tracking daemon further accepts data destined for the tracking data store from all subsystems.

[000246] The tracking daemon formats its received data into properly formatted SQL statements for inserting tracking data into the tracking data store. The tracking daemon queues tracking data in a tracking data cache to allow batch tracking data inserts into the tracking data store. The tracking daemon validates incoming data (range checks, format, etc.). Data checking is intended for internal error checking. Invalid data will be logged via the logging facility.

[000247] The tracking daemon reports errors via the Logging Subsystem and the SNMP monitoring station with an appropriate level message. In the case of data store failure or non-response, the tracking daemon queues tracking data in a tracking data cache, block all tracking calls from returning control to the application and send the error to standard error and the SNMP

monitoring station with an Emergency message level. The tracking daemon reports erroneous data inputs via the Logging Subsystem.

[000248] The tracking daemon submits tracking data entries to the tracking data store. In the case of an error, the tracking daemon sends error messages to the Logging Subsystem and the SNMP monitoring station with an Emergency message level.

[000249] **Data Upload Subsystem**

[000250] A Data Upload Subsystem is provided for receiving and processing uploaded data from clients. The input process may be made available 24x7 and may run on more than one machine. Electronic submission may be made with HTTP/HTTPS, SMTP, or FTP, should be stored on an NFS-mounted partition and should be lock-free. MAILDIR format should be used. The submission of file decompression will create a single uncompressed file.

[000251] Upon successful reception of incoming data, a unique submission ID is assigned. Receipt of data is tracked. Each submission is preferably stored in a directory named for the client-employee who transmitted the submission. A unique filename is assigned to a new submission. The Data Upload Subsystem will obtain all required configuration information from the Configuration Database.

[000252] Data which is received by the Data Upload Subsystem includes desired delivery time for data, including time zone, the priority for the groups of e-mails (e.g., bulk, normal and expedited), the Job-ID (provided by the client) for each submission. Submissions without a unique job-id or with a duplicate job-id will be rejected, client information including a valid client authorization code and optional digital signature.

[000253] The content and formatting data in the input stream preferably consists of one of the following, in the following order of appearance:

[000254] An XML body with XSL formatting information, as in the following example:

[000255] <content>

[000256] <body>

[000257] <!--- Not used until future XSL compliant --->

[000258] <!--- release --->

[000259] </body>

[000260] </content>

[000261] An empty body with DA-specific templates containing body. There is preferably provided a DA-specific body-format combination for every DA option specified, as in the following example:

[000262] <content>

[000263] <template datype=www>

[000264] <title>Ed McMahon says <var: susbt-name=name>

[000265] is a winner</title>

[000266] </template>

[000267] <template datype=e-mail daopt=html>

[000268] <title>Ed McMahon says <var: susbt-name=name>

[000269] is a winner</title>

[000270] </template>

[000271] <template datype=pager>

[000272] <var: subst-name=name> is a winner.

[000273] </template>

[000274] <template datype=IM daopt=ICQ>

[000275] <var: subst-name=name> is a winner.

[000276] See <var: subst-name=URL>

[000277] </template>

[000278] </content>

[000279] The submission metadata preferably comprises the following information:

[000280] • Expiration Date & Time

[000281] • Delivery Goal Date & Time

[000282] • Default Priority

[000283] • Number of Recipients

[000284] Global substitution data conforms to the specification as supplied in the following example of recipient substitution data. Global variable values are used during content variable substitution in cases when recipient-specific substitution value is provided.

[000285] <dadata type=IM>

[000286] <attr name=URL val=RangerURL>

[000287] </dadata>

[000288] The recipient information preferably includes the following data elements and may contain multiple instances of the following information:

[000289] a) **DA Type:** Delivery Agent Type—this is preferably either e-mail, pager or Instant Message (IM). The system of the invention may alternatively provide for delivery over a combination of multiple delivery paths, including but not limited to, e-mail, instant message, and web-based.

[000290] b) **DA Option:** The Delivery Agent Option is specific for the DA Type and may be contain the following options:

[000291] • E-mail:ASCII, HTML, AOL, MIME

[000292] • IM: AOL, Yahoo, ICQ, MSN

[000293] • Pager, via e-mail gateway, or native SMS.

[000294] c) **DA Preference:** Delivery Agent Preference will be numeric representing the order to attempt delivery by that mechanism.

[000295] d) **DA-Specific Address:** Delivery Agent Specific Address should contain information on how to deliver the message for the DA Type specified. The information may be represented as follows and conform to a DTD representing this data specified by Ranger:

[000296] • E-mail:RFC-822 compliant address

[000297] • Pager: RFC-822 compliant address or

[000298] Phone Number, PIN, and Paging System Information or

[000299] HTTP

[000300] • Postal: Postal Standard Address-Format Addresses

[000301] • Fax: Phone Number

[000302] • Voice: Phone Number

[000303] e) **DA Preference:** Only one DA Preference of any ranking is allowed per recipient.

[000304] f) **Recipient Substitution Data:** Data may be provided on a per-recipient basis for substitution into the message templates. Every substitution variable in the content must be defined by a substitution name value pair for each recipient or globally, or the recipient specific message will be rejected.

[000305] The following example illustrates the correct ordering of Recipient Information data elements.

[000306] <recipient id=xxxxxxx>

[000307] <da-data>

[000308] <da rank=1>

[000309] <addr datype=e-mail daopt=html>

[000310] <address>nobody@nowhere.com

[000311] </address>

[000312] </addr>

[000313] </da>

[000314] <da rank=2>

[000315] <addr type=postal daopt=fedex-overnight>

[000316] <street1>xxxx</street1>

[000317] <street2>xxxx</street2>

[000318] <city>xxxx</city>

[000319] <state>xxxx</state>

[000320] .....

[000321] </addr>

[000322] </da>

[000323] <substdata>

[000324] <attr name=account val=12.34.5 />

[000325] <attr name=balance val=$1.23 />

[000326] </substdata>

[000327] </recipient>

[000328] MD5 checksums will be provided for the following sections:

[000329] • Recipient

[000330] • Content

[000331] • Submission metadata

[000332] The Input Subsystem provides positive/negative feedback on acceptance of input.

[000333] The Data Upload Subsystem stores input data in MAILDIR format to a Network File System (NFS) for processing. The Data Upload Subsystem sends a message to the Parse, Validate & Load Subsystem that content has been received for delivery.

[000334] A Parse, Validate & Load Subsystem for taking the client input (a submission), parsing it, validating it, loading the data into a database, and notifying the Submission Router that it is ready for processing. The Parse, Validate & Load Subsystem accepts input from the Data Upload Subsystem. Data will is preferably contained in a single compressed file. The submission template table data may be stored as XML/XSL. An XSL style sheet is stored in the database as CLOBS. The data provided is preferably in one XML document. The following information should be supplied in the order delineated in the Input Requirements below for every unique submission to the Ranger system.

[000335] A client-unique, client-generated job-id and any optional attributes needed for submission data are included as metadata. Submissions without a unique job-id or with a duplicate job-id are rejected. The client information consists of a valid client authorization code and optional digital signature.

[000336] With respect to processing, the Parse, Validate & Load Subsystem is first notified by the Input Subsystem that a new submission has arrived for processing. The Parse, Validate & Load Subsystem then decompresses the received submission data file. The Parse, Validate & Load Subsystem then assigns a client-unique Submission ID to each validated submission. The

Parse, Validate & Load Subsystem then opens the submission file and performs the following actions:

[000337] ▪ Read client id

[000338] ▪ Validate client id against the client tables

[000339] ▪ Validate that the submission was located in the proper directory for the submitter.

[000340] ▪ Validate that the job-id is unique.

[000341] ▪ Perform additional idempotency checks

[000342] ▪ Verify the component checksums are correct

[000343] ▪ Ensure that the XML submission components are well-formed and valid according to the DTD

[000344] Validation checks on recipient address fields for all delivery types. Rejected fields will be recorded in a failure table and processing will continue.

[000345] Failure of any of the above actions will result in the following:

[000346] • Halt of processing

[000347] • Logging of error(s)

[000348] • Notification e-mail to the client of the problems with the submission

[000349] The customer's e-mail address is checked to ensure rfc822-compliant e-mail addresses.

[000350] The Parse, Validate & Load component updates submission database status column to reflect current status (valid & ready for load / rejected corrupt / rejected mal-formed, etc). In the case of successful validation, the client will be notified with the following information:

[000351] • "Your submission has been received"

[000352] • Message size

[000353] • Customer ID

[000354] • Job-ID (To be provided by the client)

[000355] • Submission-ID

[000356] • Checksum status

[000357] • Accept/reject/statistical status

[000358] • Scheduled assembly time

[000359] In the case of a failed validation the client is notified with the following information:

[000360] • "Your submission has been received"

[000361] • Message size

[000362] • Customer #

[000363] • Job-ID

[000364] • Submission-ID

[000365] • Checksum status

[000366] • Accept/reject/statistical status

[000367] • Error(s) found

[000368] The Parse, Validate & Load Subsystem inserts parsed submission data into the Submission Database, and then notifies the Submission Router of submission ready to be processed and scheduled for delivery. The Parse, Validate & Load Subsystem also stores the unparsed XML file in an Archive Database along with ClientID and SubmissionID data.

[000369] **Submission Router**

[000370] A Submission Router subsystem is preferably provided and is responsible for deciding which Delivery Scheduler will schedule processing for a particular submission. This decision is stored in a cache for future reference and in the submission database for future reference by other Submission Routers. The Submission Router communicates with the selected Delivery Scheduler and informs that Delivery Scheduler of it's responsibility for that submission. The Submission Router monitors the status (up, down and how busy) of the Delivery Scheduler to which it has assigned submissions. If a given Delivery Scheduler should become unavailable or otherwise fail, the Submission Router assigns another Delivery Scheduler to take over the process scheduling of the submission assigned to the failed Delivery Scheduler. Also in this case, the Submission Router will update it's cache and the submission database with the new Delivery Scheduler assignment information.

[000371] The Submission Router responds to requests from subsystems for information on which Delivery Scheduler has been assigned which submission. The Submission Router first checks it's own cache for the needed information. If the requested information is not found, the Submission Router queries the submission database for the needed information.

[000372] The Submission Router handles submitted data from the Out of Band subsystem relating incoming data to the appropriate submission and client information submitting this information to the tracking daemon for storage in the tracking database.

[000373] The Submission Router accepts data from a recipient's message handlers and/or domains indicating whether a message has been successfully received.

[000374] With respect to inputs, the Submission Router accepts submission-ready-for-processing information from the Parse and Validate Subsystem. The Submission Router accepts

message status data and forward it to the Tracking Subsystem for all Delivery Agents. The Submission Router accepts status information from the Delivery Scheduler to which it has assigned submissions. The Submission Router accepts requests for status information from the Administrative Subsystem. The Submission Router accepts requests for Delivery Scheduler information from all Subsystems. The Submission Router accepts requested data from the Submission Database.

[000375] The Submission Router processes incoming data from the Out of Band and Web Tracking Agents subsystems. Received data is related to the appropriate message, submission and client and submitted to the Delivery Scheduler, which submits the data to the Tracking Subsystem for storage. The Submission Router requests, accepts, and processes data from the submission database as needed.

[000376] All code in the Submission Router preferably makes use of two Global Variables:

[000377] • GlobalDebugLevel: This is used to determine the debug level at which the system is currently running. It enables libraries to use the same debug level without providing additional function calls.

[000378] • GlobalShutdown: This is used by all threads to determine if it is shutdown time. This facilitates processing, since there is no good signaling mechanism between threads.

[000379] The Submission Router sends submission information to a selected Delivery Scheduler. The Submission Router also sends requests for data to the submission database.

[000380] The Submission Router accepts a failure message for each instance in a submission where a message cannot be delivered to the intended recipient, a/k/a, a "bounced" message. Sources for the failure message may be any of the Delivery Agent Subsystems or the Out of Band (OOB) Subsystem

[000381] The failure message for each bounced message may include the following elements:

[000382] a) DATESTAMP-UTC date & time in UTC format

[000383] b) RecipientAddress address of intended recipient (address of failed delivery), could be e-mail, IM, pager, fax, etc.

[000384] c) BounceError error condition causing bounce

[000385] d) Action action taken in response to bounce, if any

[000386] e) Reason why the action was taken

[000387] ExtraComments

[000388] Delivery Agent

[000389] h) MessageID Ranger-assigned message ID

[000390] i) SubmissionID Ranger-assigned submission ID

[000391] j) RecipientID Ranger-assigned recipient ID

[000392] k) SubsystemID identifier of subsystem originating entry (EDA, IMDA, OOB, etc.)

[000393] An example of such a failure message is the following:

[000394] 571 nosuchuser@nosuchdomain.com we do not relay

[000395] The Submission Router will provide requested status information to the Admin Subsystem.

[000396] Based on return information from a recipient's message server and/or domain, the Submission Router will inform the Delivery Scheduler of the success or failure of its delivery attempts.

[000397] Tracking data is sent to the Tracking Subsystem via the Delivery Scheduler.

[000398] **Delivery Scheduler**

[000399] Delivery Schedulers are provided for processing submissions received from a Submission Router. The Delivery Scheduler initiates assembly and delivery as required. The Delivery Scheduler keeps track of the assembly status for each message of a particular submission that is scheduled for future delivery. The Delivery Scheduler preferably processes submissions immediately upon receipt from a submission router. The Delivery Scheduler handles an assigned submission in its entirety.

[000400] When the Delivery Scheduler is notified by a submission router that it is responsible for a submission, it will spawn a new thread, retrieve the necessary submission data from the submission database, build it's own internal data structures and initiate communication with an Assembly Engine. The Delivery Scheduler will be notified by the Submission Router when a recipient has received their message and no more attempts at delivery should be made. The Delivery Scheduler is also notified by the Submission Router when delivery of a message to a recipient has failed. The Delivery Scheduler will then attempt delivery via another delivery method if one is available.

[000401] The Delivery Scheduler is able to retrieve desired alternative delivery means for the recipient. That is, the Delivery Scheduler retrieves the next Delivery Agent type from the list of prioritized, alternate addresses/delivery methods specified in the submitted data for this recipient.

[000402] The Delivery Scheduler maintains message status in a database. In this respect, should the Delivery Scheduler fail, another Delivery Scheduler process can pick up processing from where the original Delivery Scheduler stopped.

[000403] The Delivery Scheduler monitors the DA Stats cache to keep track of domains that are not currently accepting messages and not attempt assembly/delivery of messages for those domains.

[000404] When the Delivery Scheduler communicates with an Assembly Engine it will tell the Assembly Engine whether the assembled message is to be passed to a DA for delivery immediately or be stored in the Assembled Messages data-store for pre-assembled message delivery at a later time.

[000405] The Delivery Scheduler will track a submission in its database whether or not a message has been pre-assembled for future delivery.

[000406] The Delivery Scheduler submits a list (size is configurable) of recipients (i.e. 1000) for which messages will need to be assembled to a given Assembly Engine. Each such assembled list is designated as a *chunk*.

[000407] The Delivery Scheduler places submissions not scheduled for immediate delivery into a special timer queue. The messages in the submission are scheduled and pre-assembled. The data structure holding the submission linked list will then be deleted. When the time comes to deliver, the submission the database will be queried and the submission linked list will be rebuilt. The actual delivery of the messages will then be initiated.

[000408] The Delivery Scheduler may be provided with functionality to divide a submission chunk into geographic chunks based on best delivery location (remote sites running the system of the invention). This division may be based upon, e.g.:

[000409] • Recipient's zip code

[000410] • Recipient's city

[000411] • Recipient's state

[000412] • Recipient's area code

[000413] The Delivery Scheduler is the intermediary between other subsystems and the Tracking Subsystem. That is, other subsystems communicate with the Tracking Subsystem through the Delivery Scheduler.

[000414] The Delivery Scheduler will accept submission information from a Submission Router. Information received will include:

[000415] • SubmissionID

[000416] • Whether the submission will be scheduled for immediate delivery or for pre-assembly for later processing

[000417] The Delivery Scheduler also accepts message update information from a Submission Router. Information received may include:

[000418] • Notification of message received by the recipient

[000419] • Notification of failed message delivery to the recipient

[000420] As part of its configuration information, the Delivery Scheduler accepts the following data from the submission database:

[000421] • host

[000422] • ports

[000423] • number of threads

[000424] • chunk size

[000425] • message cache flush timing

[000426] The Delivery Scheduler must accept a message from the Assembly Engine, which notifies the assembly scheduler that some messages were not assembled for a given batch.

The communication is preferably synchronous, and does not return until the Assembly Engine has handled all the submitted messages and has returned a status (success/failure/queued).

[000427] The Delivery Scheduler also accepts input from the DA Stats Cache, and accepts data from Web Tracking Agents.

[000428] The Delivery Scheduler sets up and stores the status of each message in a submission. The data should be cached and periodically (timing will be configurable) dumped to a database.

[000429] Upon receipt of a submission to process, the Delivery Scheduler will spawn a new thread, retrieve all submission information from the submission database and build it's own internal data structures.

[000430] When the Delivery Scheduler receives notification from the Submission Router that a message has been received, the Delivery Scheduler updates its records and database entries accordingly.

[000431] When the Deliver Scheduler receives notification from the Submission Router that a message delivery attempt has failed the Delivery Scheduler will collect the information necessary to attempt delivery via another delivery method. This relates to the "Rollover" feature, discussed in further detail elsewhere herein.

[000432] The Delivery Scheduler maintains message status in a database allowing another Delivery Scheduler to pick up processing where the original Delivery Scheduler left off should it fail.

[000433] The Delivery Scheduler monitors the DA Stats cache to keep track of domains that are not currently accepting messages.

[000434] The Delivery Scheduler tracks whether or not a message has been pre-assembled for future delivery. This information is stored in a database for possible processing by another Delivery Scheduler.

[000435] The Delivery Scheduler places submissions not scheduled for immediate delivery into a special timer queue for future processing. Messages scheduled for future processing will be assigned to an assembly engine for pre-assembly.

[000436] The Delivery Scheduler responds (ACK/NAK) to Submission Router inputs.

[000437] The Delivery Scheduler requests submission data (based on received

[000438] submissionID) from the Submission Database.

[000439] The Delivery Scheduler sends messages to a selected Assembly Engine for assembly. The information preferably includes:

[000440] • A flag to tell the assembly engine whether or not to immediately attempt message delivery or to store the assemble message in the Assembled Message data store for future delivery.

[000441] • A list of recipients and message information for those recipients for messages to be assembled. The size of the list will be configurable.

[000442] Data elements to be included in this information are:

[000443] • SubmissionID

[000444] • RecipientIDs

[000445] • Delivery Methods

[000446] • Delivery schedule (Immediate or Delayed to specified time)

[000447] **Assembly Engine Subsystem**

[000448] An Assembly Engine is provided for assembling messages as assigned by a Delivery Scheduler for either immediate delivery or for assembly for future delivery. The Assembly Engine uses a message template, message style sheet, and recipient information to assemble a customized message specifically for the assigned recipient.

[000449] The Delivery Scheduler detailing assignments to an Assembly Engine tells the Assembly Engine to either build the messages for immediate delivery or to store in the Assembled Message data store. If the messages are to be built for immediate delivery the assigning Delivery Scheduler will tell the Assembly Engine which DA to pass the assembled messages to.

[000450] The Delivery Scheduler detailing assignments to an Assembly Engine will tell the Assembly Engine if the messages assigned for processing have been pre-assembled. If the message has been pre-assembled the Assembly Engine will read the message from the Assembled Message data store. The Delivery Scheduler detailing assignments to an Assembly Engine will pass a list of messages for assembly. The Assembly Engine will retrieve the messages template, message style sheet and recipient information from the Submission Database.

[000451] The Assembly Engine is limited to variable substitution, not content creation from distinct categories. There is no limit on the number of substituted variables, or the size of the substituted variables. Variables may not be recursively substituted.

[000452] The Assembly Engine creates messages formatted for each recipient based on their DA preferences.

[000453] Should a decision be made by the Delivery Scheduler to rollover delivery to another delivery method for a particular message, the Assembly Engine will rebuild the message as appropriate for the new delivery method based on recipient preferences.

[000454] Rollover is the process in which a message is regenerated and scheduled for delivery utilizing an alternate DA, if one exists, when delivery utilizing a more-preferred DA has not succeeded. Reasons for unsuccessful delivery include:

[000455] • Failure to deliver the message using the preferred DA

[000456] • Expiration arrived on preferred DA

[000457] The DAs which are used for transmitting messages to a particular recipient can be analyzed on a recipient basis to determine which are statistically most successful. Once this is determined for a recipient, it future delivery agent choices can be influenced based on past performance.

[000458] The Assembly Engine may support the following delivery formats:

[000459] • E-mail/ASCII

[000460] • E-Mail/MIME alternate (text/plain; text/html)

[000461] • E-mail/HTML

[000462] • E-mail/AOL

[000463] • E-mail/Pager (Pager/RFC822)

[000464] • IM/AOL (TOC protocol)

[000465] • IM/ICQ

[000466] • IM/Yahoo

[000467] • IM/MSN

[000468] The Assembly Engine is preferably configured not to delay submissions due to any behavior on the part of other submissions (e.g. very long assembly time, down domains, etc.).

[000469] The number of concurrent Assembly Engines is only limited by resource availability. When a resource limit has been reached, no more Assembly Engines will be started, and an SNMP trap will be generated.

[000470] The Assembly Engine merges the components of a message from global and recipient submission data into a message prepared and formatted for delivery using XSL. The assembly engine is preferably able to generate the multi-part mime and base-64 encoding of messages.

[000471] The Assembly Engine preferably delivers assembled messages to the DA via QMQP+ based on message priority if the message should be immediately delivered. If the primary DA host is unavailable or the message should be held for later delivery, then the message will be stored in the Assembled Message data store. If the message is not assembled due to it's domain mail server(s) being down the Assembly Engine will so notify the Delivery Scheduler.

[000472] After successful assembly, the status field for the message in the Submission Database is updated to "Assembled" in the message table of the database.

[000473] The Assembly Engine reports all message tracking information to the Delivery Scheduler. Tracking status will include, but not be limited to, the following categories:

[000474] • Received for assembly

[000475] • Assembled

[000476] • Not Assembled

[000477] • Reason Not Assembled

[000478] • Sent via DA/Delivered

[000479] • Sent via DA/Queued

[000480] • Sent via DA/Rejected

[000481] • Sent to Assembled Message data store

[000482] Messages assembled by the Assembly Engine contain subscribe/unsubscribe information provided by each Ranger client. This information may be used in a normal variable substitution. Subscribe/unsubscribe information should be present in every message delivered by the system of the invention.

[000483] The Assembly Engine is preferably able to convert XML to HTML, ASCII, AOL, or any other supported format as needed.

[000484] The Assembly Engine receives the following input from a Delivery Scheduler:

[000485] • MessageID

[000486] • SubmissionID

[000487] • RecipientIDs

[000488] • Delivery_Type Immediate/Pre-Assemble - Tells the Assembly Engine whether or not the messages assigned are to be passed immediately to a DA for delivery or to the Assembled Message data store.

[000489] • PreAssembled True/False - Tells the Assembly Engine whether or not the messages assigned have already been assembled

[000490] The Assembly Engine receives pre-assembled messages from the Assembled Message data store; receives input from the Domain Stats Cache; and receives message assembly data from the Submission Database. The Assembly Engine also requests and receives blacklisted

addresses from the Blacklist. The Assembly Engine will refrain from building messages for recipients whose addresses and/or domains appear in the Blacklist.

[000491] The Assembly Engine assembles customized messages using a message template, message style sheet and recipient information.

[000492] The Assembly Engine will include Time Zone Lookup functionality such that it can determine a recipient's Time Zone from standard US Postal Service Zip Code data.

[000493] The Assembly Engine sends assembled messages to the assigned (by the Delivery Scheduler) Delivery Agent for messages delivery. The Assembly Engine should supply the VERP and Expiration Date to the Delivery Agent.

[000494] The Assembly Engine further sends assembled messages to the Assembled Message data store as directed by the assigning Delivery Scheduler.

[000495] The Assembly Engine requests message assembly data from the Submission Database based on information received from the assigning Delivery Scheduler. Those data include, but are not limited to, the following:

[000496] • Content

[000497] • DA

[000498] • Specific XSL Formatting Information

[000499] • Global substitution data

[000500] • Recipient preferences

[000501] • Recipient substitution data

[000502] The Assembly Engine output to the Delivery Scheduler includes location or time zone information to enable the Delivery Scheduler to calculate the appropriate delivery time for a given recipient.

[000503] The Assembly Engine outputs to the Tracking Subsystem a list of blacklisted recipients for whom messages were not built.

[000504] **DELIVERY AGENT SUBSYSTEMS**

[000505] **E-Mail Delivery Agent Subsystem**

[000506] A collection of E-Mail Delivery Agent Substems (EDAs) are provided for delivering large volumes of e-mail messages (e.g., in excess of million e-mail messages) per hour. With an estimated size of 15KB/message, the system requires approximately 42 Mbps outbound bandwidth.

[000507] The EDA will attempt immediate delivery and queue the message for further attempts if immediate delivery is not possible but no hard failure is encountered.

[000508] An EDA preferably comprises the following elements:

[000509] • Mail Transport Agent

[000510] • Local mail queue

[000511] • Mechanism to access remote domain mailers' readiness status

[000512] • Mechanism to update the tracking database via the Submission Router

[000513] The EDAs should be able to accommodate messages of varying priorities. This can be done on a per-EDA submission, or could be done by having the Delivery Scheduler/Assembly Engine send messages to machines, which are specific to various queue priorities.

[000514] EDAs receive messages via network connections from the Assembly Engine for immediate delivery, as well as for delivery of pre-assembled messages; the goal is to reduce disk transfer to speed delivery where possible. This may be done via QMQP+.

[000515] Each physical EDA machine will have one or more EDA processes with multiple delivery threads; the number of processes and threads is determined empirically to find the fastest combination; each process must listen on a unique socket number for incoming connections.

[000516] Each machine running an EDA process preferably has the Domain stats cache available in memory. Each EDA process will in turn have many EDA delivery threads, which will access that information from memory. This requires that each EDA machine have a separate thread running to fetch stats cache data from the database and load the memory resource; this process will use a "double buffering" approach to store data into a write-only portion while EDA processes access the read-only portion; a mutex is used to indicate which portion is readable and which is write-able.

[000517] Each EDA gets content information in its final format in order to reduce processing requirements on the EDA; this includes message headers (including VERP and Expiration), message body, variables substituted, and formatted as ASCII, HTML, or multi-part MIME as appropriate to the recipient's profile.

[000518] The DA uses Variable Envelope Return Paths (VERPs) to set a message-unique and recipient-unique envelope "From" field; this will be used by "OOB Handlers" to determine the actual recipient if mail bounces back from remote site. It will also be used by Web Agents for tracking.

[000519] For each hard-rejected message, the EDA will send a delivery failure message to the Submission Router and/or the Assembly Engine, so that it can direct the appropriate Delivery Scheduler to identify an alternative delivery means for the recipient, if an alternative delivery means has been specified in the submission.

[000520] An EDA receives messages from an assembly engine via QMQP+ protocol. Messages are preferably transmitted to EDAs via a Load Balancer to ensure the receiving EDA can accept the transmission. Once a message is received from an Assembly Engine its life should be tracked.

[000521] Each EDA will get the address in its normal "user@dom.ain" format. The EDA will attempt to resolve this domain to an available IP address by calling the Domain stats. If the domain is not in the DNS cache, the EDA will have to attempt domain name resolution to the best MX or A record itself.

[000522] EDAs receive intelligence about the availability and speed of remote domain mail transport agents from the Domain Stats cache.

[000523] The EDA will attempt immediate delivery unless the "Domain Stats" entry for the target has marked it "Down" or "Administratively Down." The EDA will first attempt immediate delivery before committing to disk, to improve performance. It should not acknowledge the QMQP+ submission until it has completed one of the following operations:

[000524] A) Successfully delivered the message and updated tracking information through the Delivery Scheduler or the Submission Router

[000525] B) Updated tracking information through the Delivery Scheduler or the Submission Router after the receipt of a hard failure (Permanent rejection by a remote MTA)

[000526] C) Committed a message to SAN disk queue and updated tracking information after the receipt of a soft failure (Transient deferral)

[000527] Mail which cannot be delivered immediately, ("Domain Stats" has marked the target "Down/AdmDown") will be queued for later delivery attempts by the EDA.

[000528] The EDA uses "Domain Stats" information to decide whether to send mail immediately or queue, and how long it should wait for remote connection and greeting, number of simultaneous queues allowed, etc.

[000529] The EDA will delete a message from it's queue when the message's expiration time passes. Expiration date/time will be indicated by a message "X-" header set by the Assembly Engine. It will also send tracking data to the Submission Router.

[000530] The EDA preferably never generates a bounce message, which might slow down other subsystems. The EDA tracks the message, but the message content of the failed deliveries should not be included.

[000531] The EDA will time the delivery and calculate the speed of connection to the remote MTA (bytes/seconds); this should be added to the tracking database for the domain.

[000532] If an EDA cannot make a connection to a target, the EDA will attempt to re-deliver the message, based on Qmail's existing exponential backoff algorithm.

[000533] The E-mail Delivery Agent updates tracking information by responding to the Assembly Engine or by sending asynchronous updates to the Submission Router.

[000534] Tracking updates are preferably as specific as possible, so as to guide subsequent attempts; these are based on SMTP reply codes in RFC-821; status values include Success, or Soft/Hard errors. Soft/Hard status will be based on the SMTP reply code, either 4yz for Transient errors (Soft) or 5yz for Permanent errors (Hard). Example database table MessageStatusKey status symbols include: EDA_250_Success, EDA_421_ServiceNotAvailable, EDA_Hard_450_MailboxUnavailable, EDA_Hard_550_MailboxUnavailable, EDA_Hard_553_MailboxNameNotAllowed. An EDASoftUnknownReason and an EDAHardUnknownReason may be needed as new ways for mail to fail are discovered.

[000535] 421 <domain> Service not available, closing transmission channel

[000536] 450 Requested mail action not taken: mailbox unavailable [e.g. mailbox busy]

[000537] 451 Requested action aborted; local error in processing

[000538] 452 Requested action not taken; insufficient system storage

[000539] 455 temporarily unable to accommodate one or more parameters [RFC-1869]

[000540] 500 Syntax error, command unrecognized

[000541] 501 Syntax error in parameters or arguments

[000542] 502 Command not implemented

[000543] 503 Bad sequence of commands

[000544] 504 Command parameter not implemented

[000545] 550 Requested action not taken: mailbox unavailable

[000546] 551 User not local; please try <forward-path>

[000547] 552 Requested mail action aborted; exceeded storage allocation

[000548] 553 Requested action not taken: mailbox name not allowed [including no-relaying]

[000549] 554 Transaction failed

[000550] 555 does not recognize or cannot implement one or more of the parameters [RFC-1869]

[000551] The EDA may need to interpret SMTP transient (4xx) errors as hard failures if the transient error would significantly slow delivery.

[000552] The EDA tracks SMTP 5xx errors to capture their numeric code and server-specific text string.

[000553] The EDA will enable an Administrator of the system of the invention to start, pause, or stop a message queue at any time.

[000554] The EDA stores the mail queue on SAN disk so the queue may be made available to other EDAs in the event of EDA failure.

[000555] The EDA updates the tracking database via the Assembly Engine or the Delivery Scheduler with success, failure or attempted delivery information, the identity of the machine that took the action, speed of the connection, and timestamp.

[000556] The EDA preferably includes functionality to deliver messages to remote MTAs.

[000557] The EDA sends requests for domain information to the Domain Stats cache/database.

[000558] The EDA will generate a failure message for each instance in a submission where a message cannot be delivered to the intended recipient, a/k/a, a "rejected" message. The EDA will pass the VERP back to the Assembly Engine or to the Submission Router.

[000559] **INSTANT MESSENGER DELIVERY SUBSYSTEM**

[000560] An Instant Messenger (IM) delivery subsystem is provided for delivering IM messages to recipients via AOL's Instant Messenger Service. The messages should be delivered quickly, and the IM delivery subsystem should allow for an early decision as to whether or not to roll delivery of the message over to another delivery method.

[000561] AOL Instant Messenger (AIM) and ICQ may both be supported. AIM uses the OSCAR protocol. ICQ uses the ICQ protocol.

[000562] FIG. 7 shows the modules that comprise the IM Subsystem.

[000563] **IM Delivery Agents**

[000564] IM Delivery Agents are provided for contacting IM servers/recipients and delivering IM messages. For each undelivered message, the IMDA will send a delivery failure message to the Delivery Scheduler, so that the Delivery Scheduler may identify an alternative delivery means for the recipient, if an alternative delivery means has been specified in the submission. The IMDAs will support 5 thousand messages per hour, collectively.

[000565] The IMDA accepts input from an assembly engine. The IMDA then sends the message to the recipient via an IM server. Input elements may include any or all of the following:

[000566] • VERP - message ID

[000567] • ImID - Screenname of the recipient

[000568] • ImMessage - URL/text

[000569] • TTL - time to live for the message

[000570] • CRC - circular redundancy check for the message

[000571] • ImType

[000572] • ClientID

[000573] • SubmissionID

[000574] • RecipientID

[000575] The IMDA will accept IM server login account information from the IM server login pool database based on InUse status (multiple logins from a single account are not allowed so if the InUse status is "TRUE" the next login account for that ImType will be needed).

[000576] • Login - login id for imType

[000577] • Password - password for login

[000578] • InUse - yes/no

[000579] The IMDA accepts module status requests and start/stop requests from the Admin module.

[000580] The IMDA accepts input from the IM Stats database. The Delivery Scheduler periodically requests IM server(s) status and associated connectivity speed to aid in determining what IM messages can/cannot be delivered (allows for early rollover decision) and what server to use.

[000581] When delivering messages to an IM server, an IMDA will request login information based on ImType from the IM Server Login Pool database and login into the assigned IM server in preparation for delivering IM messages. If the InUse status is "IN USE" the IMDA will request another login for ImType.

[000582] The IMDA reports module status when requested. Status information includes, e.g., the following data:

[000583] • Up/down

[000584] • Messages scheduled

[000585] • Threads running

[000586] • Servers connected to

[000587] An IMDA reports to a IM Server Login Pool Database when use of account information is complete.

[000588] The IMDA reports delivery-to-server success/failure to the Submission Router or the Assembly Engine.

[000589] The IMDA updates InUse based on input from an IM delivery agent when the delivery agent is finished using the account.

[000590] The IMDA requests IM server(s) status and connectivity speeds for ImType from the IM Stats data store.

[000591] If a message's time to live has expired, it will not attempt delivery of the message and will report the message as not delivered due to delivery time expiration.

[000592] The IMDA reports module status when requested, including:

[000593] • Up/down

[000594] • Messages scheduled

[000595] If a message is undeliverable, the IMDA reports delivery failure to the Submission Router.

[000596] The IMDA sends login and password information to an IM server.

[000597] The IMDA will send message(s) to recipient. Such messages include, e.g., the following data:

[000598] ImID

[000599] ImMessage

[000600] The IMDA sends message tracking data to the Submission Router or the Assembly Engine. Such data includes, e.g.:

[000601] • VERP

[000602] • IMID

[000603] • IM type

[000604] • Date-time-group if available

[000605] • Speed of user's connectivity

[000606] • TTL

[000607] • URL/message

[000608] • Delivered/not delivered

[000609] • Delivery class

[000610] The IMDA sends account-no-longer-needed status to the IM Server Login Pool database; sends module status to Admin Subsystem; updates the IM Server Login Pool database when IM login account information is released; sends "In Use" data to the IM Server Login Pool database; and sends ACK/NAK or data to Admin Subsystem in response to requests as appropriate.

[000611] The IMDA generates a failure message for each instance in a submission where a message cannot be delivered to the intended recipient, a/k/a, a "rejected" message. This can be done using the same synchronous mechanism used by the Electronic Mail delivery Agent (EDA).

[000612] The failure message for each rejected message may include the following elements:

[000613]	a)	DATESTAMP-UTC	date & time in UTC format
[000614]	b)	RecipientAddress	address of intended recipient (address of failed delivery), could be e-mail, IM, pager, fax, etc.
[000615]	c)	BounceError	error condition causing bounce
[000616]	d)	Action	action taken in response to bounce, if any
[000617]	e)	Reason	why the action was taken
[000618]	f)	Delivery Agent	
[000619]	g)	MessageID	Ranger-assigned message ID
[000620]	h)	SubmissionID	Ranger-assigned submission ID
[000621]	i)	RecipientID	Ranger-assigned recipient ID

[000622] j) SubsystemID identifier of subsystem originating entry (EDA, IMDA, OOB, etc.)

[000623] The IMDA may store data on IM server status that will facilitate the delivery of IM messages and decisions, from an IM standpoint, regarding delivery method roll.

[000624] The IMDA may be configured to accept the following data from IM server probe(s):

[000625] • ImType

[000626] • ServerID

[000627] • ServerIP

[000628] • ServerStatus

[000629] • RTT

[000630] • ServerProbeTime - date time group of last probe to this server

[000631] The IMDA may be configured to accept requests for the following server data from the IM Delivery Scheduler:

[000632] • ImType

[000633] • ServerID

[000634] • ServerIP

[000635] • ServerStatus

[000636] • RTT

[000637] • ServerProbeTime

[000638] The IMDA preferably responds to requests for server availability information from the Delivery Scheduler.

[000639] An IM server database is provided for storing data on known IM servers. The IM Server Database will accept IM server data input from the Administration module. These data will identify IM servers by type so that IMDAs will know about the server. Such data include, e.g.:

[000640] • ImType

[000641] • ServerID

[000642] • ServerIP

[000643] The IM Server Database responds to requests for listings of known IM servers by ImType.

[000644] **WEB DELIVERY AGENTS**

[000645] Web Delivery Agents (WAGs) are provided for delivering message content to a recipient via the recipient's web browser. Each WAG preferably includes the following elements:

[000646] • Web Server

[000647] • Dynamic page-generation logic

[000648] • Database access functions

[000649] Each WAG preferably handles the following types of http requests:

[000650] • 'invisible' image requests used for tracking HTML message openings

[000651] • 'click-throughs' for content not hosted by Ranger from URLs embedded in HTML messages

[000652] • user requests for content (page view requests from HTML messages) from Instant Message clients

[000653] Upon receipt of any request, a WAG will capture & record the following client request variables:

[000654] • HTTP_REFERER

[000655] • REQUEST_METHOD

[000656] • QUERY_STRING

[000657] • REMOTE_USER

[000658] • HTTP_USER_AGENT

[000659] The WAG will extract the following information from a banner ad URL:

[000660] • LinkID

[000661] • MessageID

[000662] The VERP will be present in the HTTP URL. All information necessary in processing and tracking requests should be available by knowing the VERP.

[000663] The WAG will decode the VERP (embedded with every page-view, invisible GIF and click-through (remote URL) request) to extract the MessageID of the outbound message from which this request originated, as well as the customer (Client) ID.

[000664] The WAG checks the Submission.Expiration Time of the message, and take one of the following actions:

[000665] • If the request is not valid, the request will be refused with a HTTP 404 error.

[000666] • If the Message has expired, a page will be served stating that the data is no longer available. Customers will provide a single static page for expired message pages. If they do not provide one a generic Ranger expiration page will be displayed.

[000667] The WAG will process page-view requests when the Submission.Expiration Time in the message table of the corresponding DB is greater than today's date and request reception time, i.e., when the Submission Expiration Time is in the future.

[000668] The WAG creates and delivers an HTML page to the requesting client browser when a request containing an un-expired MessageID for a valid CustomerID is received by any of the following methods:

[000669] A) Querying the corresponding database for the following information:

[000670] • Submission.XML-content.

[000671] • Substitution.Var-name 0-n occurrences

[000672] • Substitution.value 0-n occurrences

[000673] • Message.xsl-template

[000674] B) Substituting the values for var-name wherever names occur in the Submission.XSL-template

[000675] C) Formatting the SubmissionContent.Data using the Message.xsl-template

[000676] The WAG will verify the LinkID, MessageID combination obtained from a banner ad URL to make sure that it is legitimate for the particular ClientID.

[000677] The WAG will embed the message signature decoded in the request back into each page and 'invisible image' as it is delivered.

[000678] WAGs preferably support non-nested HTML tables. WAGs preferably further support the following types of ads:

[000679] • banner ads up to 400 * 200 pixels

[000680] • anchor or spot ads of up to 200 * 120 pixels.

[000681] For messages with a Submission Expiration Time greater than the current time and date (i.e., the expiration time is in the future), the WAG will provide page request and read acknowledgements to the Delivery Router, which will pass the data to the Tracking Subsystem.. The WAG will not provide this information for page requests and read acknowledgements when the Submission Expiration Time has already passed.

[000682] For messages with a Submission Expiration Time greater than the current time and date (i.e., the expiration time is in the future), the WAG may send the click-through information to the Tracking Subsystem. At a minimum, the following information should be included:

[000683] • VERP

[000684] • Link-ID

[000685] • Any information the Ranger system encodes on the URL

[000686] • Date and Time Stamp of the click

[000687] • Type of request

[000688] The WAG should not provide this information for page requests and read acknowledgements when the Submission Expiration Time has already passed.

[000689] **PROBE SUBSYSTEMS**

[000690] **MTA PROBE SUBSYSTEM**

[000691] An MTA Probe (MTAP) is preferably provided for calculating delivery time statistics. The probe gathers the set of target domains from the domain database table; this may be a list of the top 1000 most popular domains. The MTAP will probe the domain's best available MX or A record servers and determine TCP connection and SMTP server greeting time.

[000692] The MTAP computes statistics from the samples and makes these available to the Assembly Engine, Email Delivery Agents, and other subsystems by storing them in the domain database table for access by the Domain Stats Cache API.

[000693] Multiple geographically disbursed MTA Probes (e.g., the probe will provide it's own identity in the statistics so we can determine which is ``closest" to the target domain MTAs) are preferably supported.

[000694] At startup, the MTAP will get the maximum_age of samples to retain, the probe_interval, number_of_threads, and mx_sample_age to employ from the configuration database table config. Also at startup, the MTAP will retrieve the list of target DNS domain names, e.g., with `` SELECT domain_name FROM domain". (This table will be populated periodically by an external process which is outside the scope of this subsystem.)

[000695] For each domain, the MTAP will resolve the corresponding DNS MX and A records. (the ``dnscache" caching DNS resolver may be used on the MTAP host for speed; it may query a full DNS server for information not in its cache). For each set of domain MX/A records, the MTAP will sort and group them by preference: lowest MX value first, then higher MX, and finally A records.

[000696] For each domain, the MTAP connects to each address in the most preferred group and measures its TCP connection speed and time to get the SMTP greeting. If none of the most preferred group of addresses are available, the MTAP will try the next preferred group, and so on until it runs out of records. If none of the addresses in any preference are available, the domain will be marked ``down". The MTAP stores the sample information into the mxsample database table.

[000697] The MTAP can cause a database trigger to be executed which will clean old samples from the mxsample table. The MTAP trigger will cause the following statistics to be recalculated in the mx table:

[000698] ▪ connect_min

[000699] ▪ connect_max,

[000700] ▪ connect_sum,

[000701] ▪ connect_average,

[000702] ▪ connect_deviation,

[000703] ▪ hello_min,

[000704] ▪ hello_max,

[000705] ▪ hello_sum

[000706] ▪ hello_average,

[000707] ▪ hello_deviation,

[000708] ▪ success_coun,

[000709] ▪ failure_count,

[000710] ▪ status_id,

[000711] ▪ updated_dt.

[000712] The status indicates whether the particular MX (or A) host is up or down.

[000713] The MTAP trigger will cause the following statistics to be recalculated in the domain table:

[000714] ▪ status_id

[000715] ▪ mxhosts_min

[000716] ▪ mxhosts_max

[000717] ▪ mxhosts_average

[000718] ▪ wentdown_dt

[000719] ▪ updated_dt.

[000720] Here, the status is the overall accessibility of the domain; it should only become ``down" if all servers at all preferences for the domain are unavailable. If a domain is detected down, the MTAP will again probe its MX hosts a few times at a smaller interval to ensure that the first probe was not an anomaly. Only after several such retry probes fail will it mark it ``down" and record the time we noticed it unavailable.

[000721] The mxsamples table stores a finite number of samples for each ``mx" host by its IP address, trimmed by a trigger and stored-procedure as new samples are entered.

[000722] Each IP address in the mx table will have its connect and hello time statistics calculated from the mxsamples entries by a trigger and stored-procedure as new samples are entered. The status of the mxaddress will be updated likewise.

[000723] The domain table will have its connect and hello average and standard deviation statistics recalculated from a trigger and stored-procedure. The status of the overall domain will be updated likewise.

[000724] The domain table will have calculated via trigger the minimum, maximum, and average number of probed MX hosts for the current collection of mxsamples.

[000725] The time all the MX hosts for a domain have been detected as unavailable (``down") will be available as domain.wentdown_dt.

[000726] The domain states include: ``up", ``down", and ``administratively down". The ``down" state means that there are no available MTAs for the specified domain at any preference

level. An ``up" domain means that there is at least one available MTA for the specified domain. The domain-probe will not change the state of a domain marked as ``administratively down".

[000727] **INSTANT MESSAGE (IM) PROBE SUBSYSTEM**

[000728] **IM Server Probe**

[000729] The IM Server Probe is provided for probing IM servers for availability and determining/gathering connectivity speed information. The IM Server Probe accepts the following input from IM Server database:

[000730] ▪ ImType

[000731] ▪ ServerID

[000732] ▪ ServerIP

[000733] The IM Server Probe will collect the following availability/accessibility and connectivity information for each IM server.

[000734] ▪ ServerStatus

[000735] ▪ RTT

[000736] ▪ ServerProbeTime

[000737] The IMP Server Probe accepts status and start/stop requests from the Administrative Subsystem. The IM Server Probe accept the following configuration information from the Configuration Database.

[000738] ▪ Probe timing

[000739] ▪ Periodicity of server probes

[000740] As set forth above, the IM Server Probe periodically probes known (from the IM Server database) IM servers for availability/accessibility status and connectivity speed

information (RTT). The IM Server Probe will insert/update the following availability and connectivity status information in the IM Stats database:

[000741] ▪ ImType

[000742] ▪ ServerID

[000743] ▪ ServerIP

[000744] ▪ ServerStatus

[000745] ▪ RTT

[000746] ▪ ServerProbeTime

[000747] The IM Server Probe will request lists of servers to probe from IM server database. Data to be requested will include the following:

[000748] ▪ IM_type

[000749] ▪ ServerID

[000750] ▪ ServerIP

[000751] The IM Server Probe will populate the IM Stats Database with the following probe results:

[000752] ▪ ImType

[000753] ▪ ServerID

[000754] ▪ ServerIP

[000755] ▪ ServerStatus

[000756] ▪ RTT

[000757] ▪ ServerProbeTime

[000758] The IM Server Probe will send subsystem status to the Administrative Subsystem when requested.

[000759] **Out of Band (OOB) Handler Subsystem**

[000760] An OOB Subsystem is provided for handling responses from remote systems that are not immediate and cannot be handled by the Delivery Agents themselves.

[000761] VERP information is the preferred information to base OOB logging on as it provides the best information. (Additional discussion of VERP may be found in Appendix A, entitled "*Memo: VERP Versus Message ID Revision: 2.0*", which is incorporated by reference as if fully recited herein.) Any bounces returned by remote MTAs are processed by the OOB Handler Subsystem. Any replies from Recipients will also be processed. The first 2K of any reply or auto-responder will be saved in the database. Any auto-responder replies will be processed, logged and discarded.

[000762] Bounce messages from Mail Transfer Agents (MTA) due to fatal and transient errors that occur when trying to deliver e-mail are processed. These typically come from mail-daemon and postmaster accounts on remote machines.

[000763] The OOB extracts VERP information if possible; extracts DSN information if possible; and extracts the "To:" header information to see if it corresponds to the customer and submission unique "From:", "Reply-To:", and "Errors-To:" headers. The OOB will attempt to parse and automatically handle replies by the recipient of the message.

[000764] The OOB will generate a failure message for each instance in a submission where a message cannot be delivered to the intended recipient, a/k/a, a "bounced" message.

[000765] The failure message for each bounced message may include the following elements:

[000766] a) DATESTAMP-UTC date & time in UTC format

[000767] b) RecipientAddress address of intended recipient (address of failed delivery), could be e-mail, IM, pager, fax, etc.

[000768] c) BounceError error condition causing bounce

[000769] d) Action action taken in response to bounce, if any

[000770] e) Reason why the action was taken

[000771] f) ExtraComments

[000772] g) Delivery Agent

[000773] h) MessageID Ranger-assigned message ID

[000774] i) SubmissionID Ranger-assigned submission ID

[000775] j) RecipientID Ranger-assigned recipient ID

[000776] k) SubsystemID identifier of subsystem originating entry (EDA, IMDA, OOB, etc.)

[000777] An example of such a failure message is the following:

[000778] 571 nosuchuser@nosuchdomain.com we do not relay

[000779] **Billing Subsystem**

[000780] A Billing Subsystem is provided for receiving, storing and calculating data necessary to charge users/clients of the system of the invention for the services provided by the system. Information about each submission and all messages is recorded and made available to the billing system on a per-submission basis.

[000781] The information in the table below should be reported to the billing system in XML format and adhere to the DTD defined for billing. The Billing System is provided with the following for each submission:

[000782] ▪ Client name

[000783] ▪ Client Employee providing the input as a submission to Ranger

[000784] ▪ The date and time of input

[000785] ▪ The date and time of assembly

[000786] ▪ The date and time that delivery started

[000787] ▪ The date and time that delivery finished

[000788] ▪ The number of recipients in the submission input per primary Delivery Agent and summation of all recipients

[000789] ▪ The number of successful message and failed message deliveries

[000790] ▪ The number of rollovers attempted, successful and failed

[000791] ▪ The average and total message volume in Megabytes.

[000792] **Administrative Subsystem**

[000793] An Administrative Subsystem is provided to allow administrators of the system to perform tasks such as user account administration and starting and stopping of subsystems. The Administrative Subsystem user interface is preferably web-based and uses SSL where possible. The Administrative Subsystem is not meant to replace Network and System monitoring, which can be done by third party network monitoring packages such as HP OpenView.

[000794] The Administrative Subsystem preferably uses a common Authentication Data Store for the Client Administration, User Input System, and Client Reporting.

[000795] Access to system user account information may be provided for both internal users and client users. Access to all subsystems and databases may be required. Such access may be direct or through a supporting process such as the Watchdog.

[000796] The user interface should allow each of the major systems to be started, stopped, and restarted. Multiple versions of each sub-system may be running for redundancy, load

balancing and scaling. It should be possible to cleanly shut down all instances of a given sub-system or specific instances of a sub-system for maintenance (or any other reason). Administrators should be able to shut down a sub-system without causing the overall system to fail.

[000797] The Administrative Subsystem should enable a System Administrator ability to shut off a message queue on the MTA Subsystem.

[000798] The Administrative Subsystem should be configured to receive information from a web client, from Sun Solaris™ commands and utilities, from the Logging Subsystem, and/or from state information stored on NFS. An Administrator should be able to view the status of each subsystem and its host machine.

[000799] An Administrator is provided with the ability to initiate an *action* on an Ranger subsystem. *Actions* include, but may not be limited to, the following:

[000800] • Stop

[000801] • Start

[000802] • Restart

[000803] • Read-Config

[000804] • Increase logging level/verboseness

[000805] • Stop logging

[000806] • Determine status/state

[000807] Administrative Subsystem output should include, but not be limited to, the success or failure for all attempted operations and detailed failure information for failures that occur. Failure information should include, but not be limited to, the following:

[000808] • Error message

[000809] • Subsystem(s) affected

[000810] • ProcessID(s) (PID)

[000811] • Database(s) involved

[000812] Log entries of all actions should include the following information:

[000813] • Date and time stamp

[000814] • Subsystem(s) altered on a per machine basis

[000815] • Processes altered on a per machine basis

[000816] • Username of individual making changes

[000817] The Administrative Subsystem includes Submission Control functionality for providing administrators with control over client submissions. Inputs to Submission Control include: user authentication data, all client tables from the database all submission tables from the database, and a persistent data store of recipient information across all submissions. An Administrator with *write* access permission is able to perform the following tasks:

[000818] • Stop the assembly of a submission

[000819] • Resume the assembly of a submission

[000820] • Prevent delivery of a submission not yet started

[000821] • Allow delivery of a submission being held

[000822] An Administrator is able to prevent delivery across all submissions to specific users and/or specific domains.

[000823] For all submission actions, whatever their state, appropriate Database tables will be updated.

[000824] The user interface of the Administrative Subsystem displays HTML-formatted output.

[000825] The Administrative Subsystem further provides Account Management functionality. The Account Management Interface gives authorized System Administrators full access to all available system and account management functions. The Client Account Management Interface will give each client access to a restricted subset of system and account management functions. Access permissions for account management function appear in the table below.

Entity/Data	Helpdesk	NOC	Accounting	System Admin	Client Admin	Client User
Accounting data	Read	Read	Read/write	Read/write	Read*	None
Client Reports	Read	Read	Read	Read/write	Read*	None
Client User data	Read	Read	Read	Read/write	Read/write *	None
Client data	Read	Read	Read	Read/write	Read*	None
Subsystem information	Read	Read	Read	Read/write	None*	None
Admin Logging	Read	Read	Read	Read	None*	None

[000826] *The Client Administrator may only view and/or modify data to manage user accounts for that specific company. The Client Administrator is not permitted to view any information about any other company or the Ranger system. In addition, the Client Administrator may not change information about the client setup and configuration.

[000827] The Administrative Subsystem further preferably includes a Client Account Management Interface which allows a designated user employed by the client to administer user accounts for that client. Each client may have one user with administrator privileges or the capability of granting and revoking privileges to/from other users for that client. A user with administrator privileges will be able to create other users with administrator privileges and users

with lesser privileges. Client Administrator privileges will apply *only* to the particular client account. That is, a client with Administrator privileges has those privileges only the Ranger activities for his/her organization. Only System Administrators will have Administrative privileges across all system files, accounts, and tasks.

[000828] Inputs to the Client Account Management Interface include user authentication data, all client tables from the database, all submission tables from the database, and a persistent data store of recipient information across all submissions.

[000829] The Client Account Management Interface may obtain its configuration information from the Configuration Database.

[000830] The Client Account Management Interface further preferably allows Client Administrators to perform user account management for that client's own users. Client Administrators have the ability to perform the following tasks:

[000831] • Create Client User accounts

[000832] • Set/Modify the following user permissions

[000833] ▪ submitting input for submissions

[000834] ▪ viewing reports

[000835] ▪ viewing billing information

[000836] ▪ viewing client global information

[000837] ▪ administering client account information

[000838] ▪ setting and modifying password

[000839] ▪ viewing a "trial" submission or message

[000840] • Delete Client User accounts

[000841] • Suspend Client User accounts

[000842] • Display current levels of access for all users registered to the client.

[000843] • Display a list of all users register to the client.

[000844] The Client Account Management Interface preferably writes out new or updated client user account information and permissions to the account data store. All access and actions via the Client Account Management interface are logged. In particular, actions that affect the operation of the system will have logging which includes date stamp, time stamp and the user information.

[000845] The Administrative interface also allows System Administrators to do user account management. In this respect, they have the ability to perform the following tasks:

[000846] • Create User accounts

[000847] • Set/Modify User Permissions

[000848] • Set/Modify User password

[000849] • Delete User accounts

[000850] • Suspend User accounts

[000851] • Display current levels of access for a user.

[000852] The Administrative Interface should provide System Administrators the ability to manage client accounts with the ability to do all the functions that a Client Administrator may do. System Administrators should have the ability to perform the following client management tasks:

[000853] • The creation of a new client.

[000854] • The creation of Client Administrator(s) for a client

[000855] • Set/modify any necessary billing or customer specific attributes needed to process submission.

[000856] ▪ Submission priority linked to client billing

[000857] ▪ Set upload method (http, ftp)

[000858] ▪ List all clients

[000859] ▪ List all users for a given client.

[000860] • Suspend a user account

[000861] • Suspend a client (and thus all users) from using the Ranger system.

[000862] • Assign and modify Permissions for user accounts

[000863] • Delete or Purge a user account from the system

[000864] • Delete or Purge a client account from the Ranger system.

[000865] **Reporting Subsystem**

[000866] A Reporting Subsystem is provided for generating reports on system status. Inputs to the Reporting Subsystem include, e.g., the Submission, Recipient and Tracking tables, Authorization and User data, Watchdog information, and Log files.

[000867] The information should be collected from each of the sub-systems and/or the various databases being used in order to report their administrative status.

[000868] The Administrative Interface should provide General Reporting that includes a summary of submissions that are in the system, who the client corresponding to the submission is, and the state of message assembly, message delivery, and errors. Selecting a given submission will bring up the detailed information on that specific submission.

[000869] **A) General View**

[000870] • # submission submitted

[000871] • # chunks a submission has been broken into

[000872] • # recipients in a submission

[000873] • # recipients per each DA type.

[000874] • # messages sent, % complete

[000875] • # Messages sent per DA type

[000876] • # messages rejected

[000877] • # messages read

[000878] • A list of all the individual submissions in a table with the above information on a per submission bases. Also, will include: submission date and time scheduled assembly date and time.

[000879] • An indication if assembly has started and % complete.

[000880] **B) Detailed Submission view**

[000881] • job id, which is customer-specified.

[000882] • submission id

[000883] • submission date and time

[000884] • submission submitter

[000885] • # chunks a submission has been broken into

[000886] • scheduled assembly time

[000887] • assembly start or not

[000888] • scheduled message delivery date and time

[000889] • message delivery started or not

[000890] • # recipients in a submission

[000891] • # recipients per each DA type.

[000892] • # messages sent, % complete

[000893] • # Messages sent per DA type, % complete

[000894] • # messages rejected, % of each DA type reject, % of all messages rejected.

[000895] • # messages read

[000896] **EXTERNAL CLIENT REPORTING TOOL**

[000897] An External Client Reporting Tool provides clients a means to monitor status of their past or current submissions, and to view statistical information on their usage and activities on the Ranger system. An External Client Reporting Tool is also provided and receives relevant data (inputs) from the relevant subsystems and databases, and displays them in Web browsers either in the form of tables (text) or visual graphics (such as pie chart, bar chart or curve).

[000898] The External Client Reporting Tool should only be accessible using standard web browsers and the Secure Socket Layer (SSL) protocol, including IE4.0 and NS4.0 or later.

[000899] The External Client Reporting Tool's users can view any reports concerning the usage and activities of the system performed by their own company/organization.

[000900] Only authenticated users should be able to access to the External Client Reporting tool. Users can should only be able to view their own data in the External Client Reporting Tool. Their own data (submissions, messages, etc) are securely protected on the server so that it can only be viewed by themselves.

[000901] The External Client Reporting Tool displays the Summary Information on the submission's status and statistic before displaying detailed status or statistic reports.

[000902] Stat Reports can be delivered to clients either on-demand on the Web, or in e-mail or by US Postal Service. Users can request e-mail report delivery either in text-only format or text-graphics format. Users can set up their preferred delivery schedule either daily, weekly, monthly or for every single submission. Users can change their US postal address or e-mail

address for report delivery. Users can opt to export data in either raw, XML, or tab-delimited formats. The External Client Reporting Tool should be extensible to facilitate any desired changes to report type, format and content.

[000903] In the Summary Report, the status summary information should include the following information:

[000904] ▪ Submission ID

[000905] ▪ Submission's Senders

[000906] ▪ Submission Subject

[000907] ▪ Submission Status

[000908] ▪ Target Delivery (date & time)

[000909] ▪ % of Completed Submission

[000910] ▪ % of Pending Submission

[000911] In the Summary Report, the statistic summary information should include the following information:

[000912] ▪ Submission ID

[000913] ▪ Submission Subject

[000914] ▪ Total No of Recipients

[000915] ▪ Total No of Successful Deliveries

[000916] ▪ Total No of Failed Deliveries

[000917] ▪ Total No of Pending Deliveries

[000918] ▪ Average Time Delivery

[000919] The data displayed in the Status Report and Statistic Report screens are extracted from the subsystems/databases.

[000920] The External Client Reporting Tool can query the system database directly. Based on the current data model used for the system, the External Client Reporting Tool will query the Ranger database and perform joins/views on the following tables:

[000921] ▪ Submission

[000922] ▪ Sbm_track

[000923] ▪ Mesg_track

[000924] ▪ Rcp_track

[000925] ▪ Delivery

[000926] ▪ Message

[000927] In the presentation panel in the Status Report screen, clicking on the header of a column will result in the display of a sorted view of the Status table (sorted) by the clicked column values (either the ascending/descending order).

[000928] In the filtering panel in the Status Report screen, the following filtering criteria are presented for filtering:

[000929] ▪ Submit date

[000930] ▪ Submit time

[000931] ▪ Status id

[000932] ▪ DA types

[000933] In the filtering panel in the Statistic Report screen, the following filtering criteria are presented for filtering:

[000934] ▪ Submit date

[000935] ▪ Submit time

[000936] The External Client Reporting Tool should support two different types of detailed reports:

[000937] ▪ Detailed Status reports (can be referred as Status Report for short)

[000938] ▪ Detailed Statistic Reports (can be referred as Statistic Report for short)

[000939] In each of both the Status Report and Statistic Report screens, there are two panels: filtering panel and presentation panel. In the presentation panel in the Status Report screen, the data should be updated for a specified interval; that interval must be available for configuration. In the presentation panel in the Statistic Report screen, the data should be updated for a specified interval; that interval must be available for configuration.

[000940] There is only one view in the presentation panel in the Status Report screen that provides the status info on the submissions which are still in process or have already been completed but less than 3 days old.

[000941] A "Submit Filters" button may be provided in the filtering panel in both the Status Report and Statistic Report screens, which will update the presentation panel upon clicking or pressing event.

[000942] In the presentation panel in the Status Report screen, there are three different status reports presented for viewing in the tabular format which can be scrolled both horizontally and vertically:

[000943] ▪ Active submission status

[000944] ▪ Completed submission status

[000945] ▪ Scheduled submission status

[000946] An active submission status report includes the following information:

[000947] ▪ Submission Id

[000948] ▪ Submission 's sender

[000949] ▪ Submit Time

[000950] ▪ Submit Date

[000951] ▪ Submission Subject

[000952] ▪ Total number of Recipients

[000953] ▪ Total number of successful deliveries

[000954] ▪ Total number of rollover deliveries

[000955] ▪ Total number of failed deliveries

[000956] ▪ Total number of pending deliveries

[000957] A completed submission status report includes the following information:

[000958] ▪ Submission Id

[000959] ▪ Submission 's sender

[000960] ▪ Submit Time

[000961] ▪ Submit Date

[000962] ▪ Submission Subject

[000963] ▪ Total number of Recipients

[000964] ▪ Total number of attempted deliveries

[000965] ▪ Total number of successful deliveries

[000966] ▪ Total number of rollover deliveries

[000967] ▪ Total number of failed deliveries

[000968] ▪ Total number of bounced deliveries

[000969] ▪ Total number of deferred deliveries

[000970] ▪ Total number of deliveries that remained unbuilt because of inability to reach recipient

[000971] ▪ Total number of deliveries to be resent

[000972] ▪ Total number of expired deliveries

[000973] A scheduled submission status report includes the following information:

[000974] ▪ Submission Id

[000975] ▪ Submission 's sender

[000976] ▪ Submit Time

[000977] ▪ Submit Date

[000978] ▪ Submission Subject

[000979] ▪ Total number of Recipients

[000980] ▪ Target Delivery (time)

[000981] ▪ Expired Time

[000982] ▪ Estimated Time Delivery

[000983] ▪ Estimated Billing costs

[000984] In the presentation panel in the Statistic Report screen, different statistical views are supported (the 1st 4 views must be supported in the first version of the system):

[000985] ▪ Percentage of completed submission versus time delivery (Time statistical view)

[000986] ▪ Percentage of completed submission versus AVERAGE time delivery (Average time statistical view)

[000987] ▪ Percentage of successful, bounced, rejected and deferred deliveries (Delivery statistical view)

[000988] ▪ Percentage of recipient who have read the message versus submission (Recipient statistical view)

[000989] ▪ Percentage of different message formats (ASCII, HTML, XML, WML, SMS, etc.) (Message statistical view)

[000990] ▪ Percentage of number of messages versus DA Type (DA statistical view)

[000991] ▪ Number of bandwidth utilized versus submission (Bandwidth statistical view)

[000992] ▪ Number of KB graphics, KB text and average message size (in KB) (HTML statistical view)

[000993] ▪ Number of click-thru clicks, ad-banner clicks versus submission Id (Click statistical view)

[000994] ▪ Percentage of SLA (Service Level Agreement) compliance versus submission (SLA statistical view)

[000995] In the presentation panel in the Statistic Report screen, the following choices are presented for selecting one or more views to be displayed:

[000996] ▪ Time statistical view

[000997] ▪ Average time statistical view

[000998] ▪ Delivery statistical view

[000999] ▪ Recipient statistical view

[0001000] ▪ Message statistical view

[0001001] ▪ Delivery Agent statistical view

[0001002] ▪ Bandwidth statistical view

[0001003] ▪ HTML statistical view

[0001004] ▪ Click statistical view

[0001005] ▪ SLA statistical view

[0001006] In each statistical view in the Statistic Report screen, data can be visually displayed in the form of a bar chart, pie chart or curve, or can be displayed as text in the tabular format.

[0001007] SYSTEM & NETWORK MONITORING

[0001008] Initially, each system should be monitored by a Network Operations Center (NOC) using, e.g., HP OpenView. Thresholds should be decided upon, and procedures developed to respond to conditions exceeding thresholds. This includes, is not limited to, the following on each machine:

[0001009] • System load

[0001010] • Memory

[0001011] • Swap use

[0001012] • Disk utilization

[0001013] • Disk performance

[0001014] • Network Utilization

[0001015] System & Network Monitoring functions will give operators, system administrators, and service administrators a view of the overall health of the system through a web Browser. Features available may include, e.g., the following:

[0001016] ▪ Watch dog process with alarms.

[0001017] ▪ Real time view of System going down or not responding available on refresh period, set as a configuration DB element.

[0001018] ▪ Accessible information should include, but not be limited to, the following:

[0001019] ▪ A list of all major sub-systems, the machine(s) that they are running on, and their state (up, down, degraded, administratively down), and the current load on those sub-systems should be viewable.

[0001020] • The EDA Statistics Cache information should be viewable sorted based on domain name or it's rank in receiving messages.

[0001021] **Watchdog Subsystem**

[0001022] A Watchdog Subsystem is preferably provided and has responsibility for monitoring all the subsystems and reporting on their application level health and status. It is not intended to replace network or host monitoring software, such as HPOV or BMC patrol.

[0001023] The Watchdog Subsystem monitors the following conditions:

[0001024] ▪ Number of available threads per subsystem instance

[0001025] ▪ Availability of each individual subsystem

[0001026] ▪ Subsystem utilization level (memory, queues, disk, ...)

[0001027] ▪ % Complete of current jobs.

[0001028] The Watchdog Subsystem will obtain its configuration information from the Configuration Database.

[0001029] **IN-MEMORY**

[0001030] **Instant Messenger Cache**

[0001031] The Instant Messenger Cache stores account information (login and password) for IM servers. Multiple login accounts will be required per ImType to allow multiple IMDAs to

deliver to a single ImType simultaneously (only one login at a time per account). This database will store whether or not a login and password combination are currently in use.

[0001032] The Instant Messenger Cache will accept IM server login information input from the Administration module:

[0001033] ▪ ImType

[0001034] ▪ ImAccountID - Ranger assigned ID that identifies this account information

[0001035] ▪ ImLogin - Screenname

[0001036] ▪ ImPassword- Password for ImLogin

[0001037] ▪ InUse - True/False, tells whether or not this account is in use

[0001038] The Instant Messenger Cache inserts data received from the Administration module into the database. It also responds to requests for account information from an IM delivery agent. When an account is made available to a delivery agent InUse is updated to reflect that the account is in use.

[0001039] The Instant Messenger Cache preferably provides the following IM account information.

[0001040] ▪ ImLogin

[0001041] ▪ ImPassword

[0001042] The Instant Messenger Cache also provides account listings and account status to the Administration module:

[0001043] ▪ ImType

[0001044] ▪ ImLogin

[0001045] ▪ ImPassword

[0001046] ▪ InUse

[0001047] **MTA DMTA Database Cache**

[0001048] The developer interface to the DA stats cache is a query utilizing a domain name with an IP address returned. The IP address returned will be determined in an efficient, load-balanced method from a set of IP addresses.

[0001049] Domain lookup code must be accessible quickly via an efficient hash. Additionally, if a domain is not found via the hash algorithm, that must indicate that the domain is not in the hash, not that that the hash algorithm missed.

[0001050] The domain cache algorithm should reload data on a configurable interval or when a SIGHUP is received by the parent process, i.e., Select IP, ConnectTo, HelloTo from DAStats, MX where DAStats.DomainID = MXDomainID and MXHelloTo <= (DAStatsHelloAvg + N * DAStatsHelloSdev) and MXStatus = 'up' and MXPrefernce = (select min(preference) from MX MX2 where MX2DomainID = MXDomainID and MX2Status='up') group by DAStatsDomainID.

[0001051] The domain cache must govern it's own timing to meet the above requirement without requiring programmer intervention.

[0001052] **SYSTEM EXTERNAL INTERFACE**

[0001053] A single web location is provided to which all system clients will connect for access to all client functions. The external interfaces supported by the system preferably include:

[0001054] • Online user access web pages for registered customers

[0001055] • Online user access web pages for administrative users

[0001056] External interfaces are provided for electronic mail and Instant Messenger.

[0001057] **SYSTEMS AND NETWORK ARCHITECTURE**

[0001058] **VRM / Load Balancing**

[0001059] Appropriate VRM/Load Balancing solutions are available from Cisco, Resonate, Radware and IPivot. Certain key discriminators may be used to determine the product(s) appropriate for the System. Those criteria include:

[0001060] ▪ Architecture

[0001061] ▪ Performance and Scale

[0001062] ▪ Reliability

[0001063] ▪ Feedback Techniques

[0001064] ▪ Redirection Methods

[0001065] ▪ Policies

[0001066] ▪ Advanced Features.

[0001067] Each of these are described, together with the specific attributes appropriate for the System, in the following paragraphs.

[0001068] Architecture

[0001069] Distributed architectures are a must, so that the Load Balancing units do not themselves become either a bottleneck or failure point. Packaging, platform and form-factor all are considerations as well. Software-only solutions, Packaged, PC-based offerings and Custom Hardware are all available. Although some very large companies are involved in this market space, none of them (with the exception of possibly Cisco) have the best products.

[0001070] The packaged solutions offer the advantage of minimal configuration effort while maintaining reasonably up-to-date feature sets, since custom hardware is not involved. Radware, Ipivot and Cisco all offer solutions of this type. Capacity upgrades per unit are not as

straightforward as with the software-only solutions. Custom ASIC-based Switching VRMs help to simplify overall configurations by combining load-balancing with a layer 2 or 3 switch, but tend to lag in features, behind the other types. Performance is mixed in this category as well, ranging from not great to arguably the very best performer of any type (from Alteon Systems).

[0001071] Performance and Scale:

[0001072]	Hits per second	36	720
[0001073]	Number of simultaneous users	50,000 now	1M in 4 yrs
[0001074]	Bandwidth	7.6MB/sec	140MB/sec

[0001075] Preferred Feedback Techniques:

[0001076] Active Content Verification (ACV) with response time monitoring should be available.

[0001077] Redirection Methods:

[0001078] DNS Re-Direction is preferred as basic, or core.

[0001079] Recommended Policies:

[0001080] Weighted Round Robin or Least "Used" (connections or CPU or RAM). QoS policies concerning moving resources between applications during peak loads are also nice to have. Any Geographically-dispersed load-balancing policy should be based on the Acuitive advice to "use a site in your region unless it's on fire."

[0001081] **System Interfaces**

[0001082] External Interfaces

[0001083] In order to support both current & new customer data requirements in a robust fashion, external interfaces are based on XML wherever possible. This includes bulk data input, as well as DA-specific data transformations (formatting in HTML, WML, IM formats, etc.). The

System may output HTML to browsers, and may also utilize data-bound XML for clients that can handle such.

[0001084] The following subsystems will support DA-specific protocols:

[0001085] ▪ EMail DA QMTP & SMTP

[0001086] ▪ IM DAICQ & OSCAR

[0001087] ▪ OOBs SMTP

[0001088] ▪ Pager DAs QMTP & SMTP

[0001089] ▪ PDA DAs QMTP & SMTP

[0001090] Internal Interfaces

[0001091] Internal system interfaces are based on the RMI, 'X' & QMQP protocols. The system may be designed to utilize RMI-IIOP should the need arise to integrate with CORBA applications (such as client 'back-end' systems). The following essential application services communicate over RMI:

[0001092] ▪ Data Input

[0001093] ▪ Validation & Load

[0001094] ▪ Submission Routing

[0001095] ▪ Delivery Scheduling

[0001096] ▪ Assembly Engines

[0001097] ▪ WAGs

[0001098] ▪ Tracking

[0001099] The following subsystems should support XML:

[0001100] ▪ Data Input

[0001101] ▪ Assembly Engine

[0001102] ▪ WAG

[0001103] The Data Input subsystem uses RMI for internal communications, but also supports XML - centric document exchanges with clients. The Assembly engines need to understand XML & perform XSLT transforms, but communicate internally over RMI & QMQP+ (to DAs).

[0001104] The following subsystems need database access either through container – managed persistence, a database connection (JDBC) and transaction management service or directly:

[0001105] ▪ Validation & Load

[0001106] ▪ Submission Routing

[0001107] ▪ Delivery Scheduling

[0001108] ▪ Assembly Engines

[0001109] ▪ WAGs

[0001110] ▪ Tracking

[0001111] Bean-managed persistence is also necessary for complex queries & update transactions flowing through EJBs.

[0001112] **PROCESS DISTRIBUTION**

[0001113] The diagram given in FIG. 8 portrays a mapping of logical processes and application layer protocols to the physical nodes comprising the System. Note that not all DB links are shown, due to diagrammatic limitations. The flow of information proceeds (roughly) from upper-left of the diagram through the lower left, then on through the center & out on the right.

[0001114] Both the Delivery Schedulers and Submission Router subsystems are shown running non clustered, on a pair of Sun 'workgroup' servers: a dual-CPU E450 and a single-CPU E250. This node specialization will enable significant throughput gains and may be augmented through judicious use of stored procedures, so that data access, parsing & formatting computations can be shared across a larger number of processors, I/O channels and controllers. Both the assembly engine and the parsing, validating Loaders ('pvLoaders') will run as EJBs hosted under the J2EE compliant WebLogic Application Server (WLS).

[0001115] WebLogic plug-ins on all web servers enables them to communicate efficiently over RMI to any components & EJBs in the application tier. Data input is run on a triplet of Netra T1s, together with the WAGs and reporting & billing subsystems, all of which is implemented with Servlet technology, utilizing the Sun MVC guidelines, time permitting. All three run Apache with the WebLogic plug-in &/or Jserv. 'Front' or 'Controller' servlets will run under Solaris JVMs on these machines. In addition, the Input subsystem also incorporate a customized set of integrated ftp daemons.

[0001116] All Delivery Agents run on T1s, as well, for maximum control of scale-group increments. Initially, 4 or 5 are allocated to the Email DAs (including EMail-gatewayed pager & PDA DAs), with two for IMs and Probes. The Email & IM DA scale groups will therefore provide some measure of redundancy as well as scaling potential.

[0001117] **NAMING & DIRECTORY SERVICES**

[0001118] **Service Location**

[0001119] JNDI & SQL

[0001120] Service lookup (& the foundation for Location transparency) is supported through a distributed service naming mechanism. For Java components with access to the WebLogic

Server (WLS) cluster, this may be accomplished through use of the WLS implementation of the Java Naming & Directory Interface (JNDI). This should provide location – independent service look-up for all Java services. The WLS JNDI supports multiple types of service directories, including LDAP, SQL, NDS, etc. For V1 Ranger, an SQL table (or set of tables) is used to support this capability. The following information should be stored in this service 'directory' regardless of implementation:

[0001121] service lookup data structures:

[0001122] ▪ services:

[0001123] ▪ service_name

[0001124] ▪ service_hostname (or ip_address)

[0001125] ▪ service_port_no

[0001126] ▪ service instances:

[0001127] ▪ instance_name

[0001128] ▪ instance_hostname (or ip_address)

[0001129] ▪ instance_port_no

[0001130] ▪ preference / priority

[0001131] ▪ service_name (FK)

[0001132] Searching the first table by service name would yield a unique (load balanced) host+port combination for client subsystems to connect to. A search by service name on the join of these two tables should yield all rows for the named service to see which instances are mapped to which services. Searches by instance name on the second table should yield a unique row (a singe host & port# combination). The first will be useful for client subsystems seeking

connections to a service, while the last would support subsystem queries for host&port combinations to which listeners must be bound.

[0001133] The same table(s) may be used to support service lookup (as well as other critical service management capabilities, such as configuration lookup) in a common fashion for both Java and C/C++ subsystems, in the following manner:

[0001134] **SQL-basis**

[0001135] Those subsystems which do not have access to the SQL database, nor to JNDI (i.e., possibly the DAs), service lookup may be supported from the same data structures, but via another subsystem, the Configuration & Service Manager (CSM). This subsystem is comprised of a set of persistent Java process(s) and a set of entity beans backed by the Oracle DB. All 'client' Java subsystems invoke methods on the remote interface of these beans to get config information, update subsystem status, and, if necessary, lookup service host & port combinations (although this last should be performed via WLS JNDI). The concept is illustrated in FIG. 9.

[0001136] C processes will need either client libraries to connect to the CSM or direct knowledge of, and access to, message formats for requesting and receiving configuration, status & service location information. They then forward the messages & requests on to the CSM daemon(s) and through the CSM EJBs to the DB. The inclusion of a library with the necessary methods/functions needed for the CSM API would enable those subsystems with neither JNDI nor DB access to still acquire service location (host address & port#) information in a centralized, standard fashion.

[0001137] Such an API should include functions of the following type:

[0001138] char * getmyHostandPort(char * myServiceInstanceName) ;

[0001139] char * getRemoteServiceHostandPort(char * remoteServiceName) ;

[0001140] To make this work, one additional piece is required, namely, a 'bootstrap' service-lookup for the CSM. This may be provided via the same config DB table via the same EJBs, or as an alternative, by configuration flat-files mounted on the NFS (Netapp) filesystem. The daemons, of course need DB connection information, but this is commonly provided through the Oracle client connectivity infrastructure (the tns_names.ora file).

[0001141] Note that standard configuration information available in the config DB table, together with system/subsystem status information (in the system_status DB table) could also be made available in the same fashion.

[0001142] A (system administration) client page is needed to allow sysAdmins to

[0001143] perform queries as well as update DB settings via the CSM beans.

[0001144] **Alternative: NFS flat-files**

[0001145] A viable alternative to the SQL basis just described is to use flat files mounted on the NetApp (NFS) filesystem for all configuration information, including service lookups. Since all nodes in the Ranger complex will have access to the NFS filesystem, all services could read the required host+port combinations from these flat file(s). Regardless of implementation, however, the information to be carried in the service-lookup data structures is essentially the same.

[0001146] If flat-files are used, it may still be useful to generate them from a central pair of DB tables, then utilize a simple query process to export this information and store in the NFS filesystem.

[0001147] **Interface Support**

[0001148] There are 4 major categories of interfaces that should be considered for the System, all of which may be supported using a combination of the Sun J2EE specification, supplemented with the XML standard.

[0001149] These 4 categories include:

[0001150] 1. Client Data input - for V1, a set of XML DTDs will be specified with sufficient flexibility to allow for client-specific data substitution & DA-specific formatting. Future versions may support ICE, or some form of XML-RPCs (WIDLs)

[0001151] 2. Inter-process communications - the JNDI & 'CSM' approach just described should be used for service location, configuration, status reporting & naming. RMI (the WebLogic optimized version wherever available) for remote invocations will allow Ranger Java components to communicate effectively. QMQP+ and/or an Ranger sockets-layer API ('X') for remoting IPCs among C processes or between C & Java processes. Future versions may upgrade to the Tuxedo 'message switch' technology.

[0001152] 3. Database Access - direct DB access will be limited to a minimal set of common DB - oriented services (e.g., tracking, AEs & DSs). These services will pool connections & provide an abstract DB interface using WLS optimized JDBC or native DB interfaces (where necessary for performance, e.g. pro*C/OCI over SQL*net)

[0001153] 4. DA-specific protocols - to support remote interactions with message facilities such as MTAs, IM servers, etc., the native protocols will need to be supported. Such support will be provided with an abstraction layer within each DA. to shield the rest of the Ranger application to the extent possible from changes in these protocols. Supported protocols will include: QMTP, SMTP,ICQ, and others.

[0001154] **Database & transaction Services**

[0001155] **Processing Model**

[0001156] Although direct client subsystem access to the System's database is minimized, the transaction processing model will resemble very closely that of a very high volume OLTP application. The clients responsible for much of the transaction volume on a constant basis will be certain of the services within the System itself, especially the following modules:

[0001157] • Parse, Validate & Load

[0001158] • Delivery Schedulers

[0001159] • Assembly Engines

[0001160] • Tracking Daemon(s)

[0001161] • WAGs

[0001162] Other components may also require direct DB access, most other services will perform the bulk of their updates/inserts through the tracking subsystem daemons. With projected volumes numbered in the 10s of K tps, a very aggressive and super-fast DB model is required. The relational model is chosen for this application as a natural fit, as well as being the state-of-the-art for all but the very most complex of entity types and relationships.

[0001163] **RDBMS vendor:**

[0001164] The most robust and fastest RDBMS in the server tier is Oracle. This DB engine has been chosen as offering the very best combination of world-class speed, robustness & reliability, as well as entity & relational integrity. Special features and tools available with 8i made this particular version the most attractive. The foundation for all database storage for the System is preferably a set of tables implemented in Oracle 8i.

[0001165] **Transaction Processing**

[0001166] DB connection pooling and 'funneling' or multiplexing will be supported via the Tuxedo or M3 tpm/OTM products from BEA systems (in future versions). The connection pooling bundled with Oracle 8i and / or the container - managed persistence and optimized JDBC classes available with the J2EE-compliant server (WebLogic) may be used.

[0001167] **Caching**

[0001168] The fastest performance can be achieved when DB operations are performed at (or usually at) memory, rather than disk speeds. For this reason, two levels of caching are preferably used. The first is provided by the local RAM on the E4500 DB server, as shown in FIG. 10.

[0001169] As much RAM as may possibly be spared on this machine should be configured for Table (and index) caching. An estimate of just over half of the total RAM can be made available for this purpose. The second level of caching will provide 'disk write spoofing' with nearly 100% guarantee of RAID disk write commit at the file-system level, so that even the RDBMS will 'see' the second cache as disk.

[0001170] Of all the database elements which may benefit from this two-layer caching, the following are top candidates:

[0001171] ▪ redo logs for active submission DBs

[0001172] ▪ tracking tables (recipient track, msg_track & sub_trk) for active submissions

[0001173] ▪ primary index files for these tables

[0001174] This caching concept may be extended further using super-high performance memory-optimized databases or DB-'front-ends' from vendors such as TimesTen to achieve transaction rates of the stated magnitude.

[0001175] **Database Organization:**

[0001176] The primary DB organization schemes available for satisfying the needs of concurrent submissions for multiple clients include the following:

[0001177] • One DB w/ one set of tables for all submissions

[0001178] • One DB with separate table spaces containing table-groups per submission

[0001179] • Multiple DBs (one per submission)

[0001180] The drawbacks to #1 include both 'perceived' security risks for individual clients and other 'trust' issues as well as super-large table cardinalities (in the hundreds of millions or even billions of rows possible). The advantage is simplicity and consolidated overhead.

[0001181] Approach #3 brings up serious resource – waste and overhead issues, with up to twenty-four DBs per day being created (or loaded) and possibly hundreds concurrently maintained (if submission lifetimes are measured in weeks.)

[0001182] The best approach may be #2, whereby all 'common' or 'persistent' tables are accessible to groups of tables 'owned' by individual submissions. the naming & access conventions for this approach would involve:

[0001183] ownerID.DBname.tablename.columnName

[0001184] **Web Services**

[0001185] As shown in FIG. 11, Apache is preferably used as the standard web server for all web components, including:

[0001186] • the Web Agents (WAGs),

[0001187] • the Data Input subsystem,

[0001188] • the System Administration pages

[0001189] • Client Reporting.

[0001190] As far as possible, Weblogic is used to manage all Servlets employed for dynamic page creation (JSP is deprecated for this project). Hence the Weblogic plug-in for each Apache instance, as shown in the figure.

[0001191] Where a budget-driven fall-back is required for Servlet management, Jserv (also from Apache, and strongly influenced by Sun's J2EE initiative) may be utilized. If necessary, a hybrid architecture incorporating both can be fielded, although the preference would be for a pure Apache – Weblogic implementation, for simplicity of configuration and maintenance. All Web nodes run on Netra T1s with .5 GB of RAM and 36 GB local disk. The OS, Web Server SW, config files, Web server logs and 'static' pages, are stored locally along with the 'Front'/'Controller' Servlet classes.

[0001192] Servlets run under the JServ Engine using Solaris JVM(s). The interfaces from the 'Front' or 'Controller' (main navigational, 'gateway') Servlet to other classes utilize standard object instantiation and method invocation. EJBs are employed using standard J2EE interfaces, as supported by the WebLogic plug-in. Thus, the application designers need only invoke methods on the 'remote' interface to each bean and not be concerned about the details of bean location or management. Typical bean life-cycles will involve one call to instantiate a 'bean-home' followed by a bean-home.create() invocation. This form of remote method invocation (RMI) is supported over the highly-optimized Weblogic RMI via the WebLogic plug-in.

[0001193] **Application Services**

[0001194] **EJB components**

[0001195] Key application-layer services run under control of the Weblogic application server, thus deriving full J2EE–compliant life-cycle management and component support, including optimized, multiplexed database connection caching and pooling, transaction

management, persistence, state maintenance, process monitoring and restart, thread management and flexible security.

[0001196] The following services run in the JVMs managed by WebLogic:

[0001197] • Assembly Engines

[0001198] • Deliver Schedulers EJBs

[0001199] • Submission Routers EJBs

[0001200] • Parsing/Validating Loaders

[0001201] These services are constructed of distributed components conforming to the EJB specification. Key APIs to be employed will include RMI, JNDI and JDBC, as shown in FIG. 12. All of these are J2EE conformant interfaces presented by highly optimized and configurable WebLogic services. At least two of the E450s will need to run instances of WebLogic with the clustering option.

[0001202] Although this diagram portrays both the WLS & the Tuxedo message switch running on each of the four 'middle tier' server nodes, this is not required, nor necessarily optimal. The 'Jolt' interface shown is a remoting IPC connection for exchanging high speed messages, so that instances of the WLS and Tuxedo message switch may be allocated as desired across machines in this tier. A minimum of two instances of each is recommended for redundancy.

[0001203] **Messaging Services: Delivery Agents (DAs)**

[0001204] For the Delivery Agents, a set of high-performance messaging servers are utilized, all based upon a similar architecture comprised of the following elements:

[0001205] C language implementation

[0001206] static libraries for common services

[0001207] support for QMQP+ for intra-system messaging & queuing

[0001208] support for:

[0001209] either: QMTP & SMTP for communication with remote

[0001210] or: ICQ & OSCAR for communication with MTAs & IM Servers

[0001211] a disk-based retry queue for temporarily holding outbound QMQP messages

[0001212] multi-threaded design

[0001213] Init, reconfig & thread management components

[0001214] This architecture is shown in FIG. 13.

[0001215] **C/C++ Services**

[0001216] The Tracking Subsystem, together with the Assembly Engines and DAs may utilize the Tuxedo message switch technology and therefore incorporate services built to the Tuxedo ATMI API. This provides a super-fast, bi-directional, multiplexed, rich socket interface. This IPC requirement will need to be met by a sockets-layer API. The retry queue will be necessary for handling QMQP messages which soft-fail on the first delivery attempt. This queue is serviced by a disk-based 'vanilla' version of the Qmail EDA.

[0001217] As shown, a threaded version of the QMQP daemon is combined with a threaded version of the delivery agent itself (formerly Qmail send, rspawn & remote). The worker threads which handle remote MTA interactions are dispatched from a pre-allocated pool upon receipt of a new message by the QMQPD listener(s). These listener(s) will support a persistent variant of QMQP called QMQP+ for accepting message flows from the AEs.

[0001218] Oracle access by the DAs may be necessary. If so, it may be used for configuration & service location lookups, etc. Normal tracking flows, however, should go through the responses to the AEs and through the SRs.

[0001219] Inter-system protocols such as QMTP & SMTP will need to be supported for EMail DAs & 'EMail-Gatewayed' Wireless DAs. The latter may differ very slightly from the EMail DAs, or be part of a common implementation. For IM DAs, the roughly corresponding protocols will be ICQ & OSCAR.

[0001220] The decision to model IM DAs after the EMail DA design is made to try to simplify overall interface complexity for the application, and in particular, the Assembly Engines. If all DAs can support just one disk-based and one 'remoting IPC' protocol for intra-system messaging, then the Assembly engine's burden of complexity is kept to a minimum.

[0001221] 'Worker Thread' pools should be created at startup so that individual threads may be detached and assigned to message processing (de-queuing & transmission) as quickly as possible. Signal-handling, (re)-configuration and initialization tasks should be provided for. As far as possible, separate components should be encapsulated in discreet functions in order to isolate them from each other and the main logic flow. Clear and distinct interfaces for each service, API and protocol should be elaborated at design time.

[0001222] The QMQP+ protocol may be replaced by a Tuxedo message switch client or other protocol 'X' candidates. The client may be implemented if desired to enhance intra-system messaging performance. Such a client would be configured to carry Intra-system Logging & Tracking messages as shown and possibly QMQP traffic as well.

[0001223] **User Interface**

[0001224] A permutation of the Sun Microsystems' Model-View-Controller (MVC) architecture should be employed as far as possible for all user-facing web pages, with the possible exception of the WAGs. JSP is deprecated, however, in favor of a set of Servlet classes

designed to handle HTML page generation, embodying methods for header, trailer and body.navigation and body.content generation.

[0001225] Controller Servlets may be used to extend the httpServlet class, while model classes may be implemented as EJBs or at least employ beans, as appropriate, for DB access.

[0001226] **BASIC I/O OPERATIONS**

[0001227] **directory organization**

[0001228] In order to keep the size of directories below operational limits, messages will be written to separate directories within directory trees of minimal depth. A variety of methods for dividing a set of messages are possible, including by message group or 'chunk'. For deferred deliveries in particular, some form of message allocation among sub-directories within a chunk may be necessary. The following guidelines are suggested:

[0001229] A directory tree whose 'root' name will contain the client identifier

[0001230] One 'parent' subdirectory within the root for each chunk of the delivery

[0001231] N subdirectories within both each chunk directory, created dynamically

[0001232] never more than K files in any directory where K is configurable at initialization time (and probably never more than a few hundred)

[0001233] as far as possible, create chunks across distinct file systems

[0001234] Distinct file systems are recommended in any case to maximize file I/O performance, even where a SAN or NFS appliance is used. Up to a point at least, the more spindles involved in a write operation, the better.

[0001235] Use of MAILDIR protocol

[0001236] The maildir protocol may be used for file storage where multiple writers and readers need to operate on a common set of files. File locking is normally required to ensure that

file-reading clients will never try to open or use incomplete files. The maildir protocol circumvents the need for file locks by declaring that all writes are performed to a directory (usually \tmp) separate from that used for reading, with a move operation from \tmp to the queuing directory after each write. Depending upon the relative performance achieved by this approach, the maildir protocol will likely be used instead.

[0001237] Use of SAN filesystems and NFS-safe mounted partitions:

[0001238] SAN storage may be utilized for message queues of all types. The NFS appliance can supply filesystems for logging files, configuration files and other common files. Local disks should only be used for storage of copies of:

[0001239] packages

[0001240] SW products

[0001241] config files.

[0001242] QMQP

For disk-based message queuing, the Quick Mail Queuing Protocol (QMQP) may be used. QMQP provides the following advantages:

[0001243] centralized mail queue within a cluster of hosts

[0001244] much faster than SMTP (Simple Mail Transfer Protocol).

[0001245] QMQP clients & servers are easier to implement than SMTP servers/clients.

[0001246] QMQP employs the NETSTRING encoding of data and uses TCP/628 transport.

[0001247] NETSTRINGS

[0001248] A netstring is a self-delimiting encoding of a string which eliminates the need for end-of-string zero-byte searching. It has the following major characteristics:

[0001249] Declares the string size up front.

[0001250] Basic building block for reliable network protocols.

[0001251] Any string of 8-bit bytes may be encoded as [len]":"[string]",".

[0001252] Example:

[0001253] String: "hello world!"

[0001254] Encoded as: <31 32 3a 68 65 6c 6c 6f 20 77 6f 72 6c 64 21 2c>,

[0001255] Alternate: "12:hello world!,".

[0001256] **FILE I/O**

[0001257] Block buffered file operations are preferred where possible for basic file input and output (I/O). This implies the use of, for C programs:

[0001258] int read(int fd, char *buf, int n) ;

[0001259] where the arguments are as normally specified, but ensuring that the 3rd argument is tuned to a multiple of the blocksize in use on the filesystem in question. This may be set to 1024 bytes. The corresponding output function is, of course, write.

[0001260] For java programs, buffered I/O is possible using the BufferedInputStream and BufferedOutputStream classes, where writes, for example, would be invoked on a BufferedOutputStream instance buffO, such as:

[0001261] buffO.write(byte[] myBuff, int offset, int len) ;

[0001262] **REMOTING IPCS: APPLICATION – LAYER MESSAGING**

[0001263] As set forth above, remote interprocess communications (remote IPC) requirements of the system may be met by a combination of:

[0001264] RMI for Servlet – EJB, Bean – Bean and Java 'daemon' - EJB invocations.

[0001265] RMI-IIOP or 'X' for Java – C communications.

[0001266] QMQP+ for Assembly Engine – DA message streams.

[0001267] Alternative candidates, for protocol 'X', include the WebLogic/Tuxedo message switch, and the Message Passing Interface (MPI), Starburst, and IP multicast protocols.

[0001268] A sockets-layer interface API may be used by subsystems which require it. Persistent TCP sockets may be utilized wherever possible. That is, connections between subsystems (or between tracking client library instances & tracking daemon(s)) will be established at start-up time (as well as upon receipt of SIG_HUP).

[0001269] Although a large number of functions are available in robust messaging libraries such as Tuxedo, the Sun MPI implementation, or IBM's MQ Series, for example, and a wide range of distributed application problems may be solved with about 6 – 8 major functions. These functions are usually of the following types:

[0001270] • communications (library & related facilities) initialization

[0001271] • communications (library & related facilities) termination & release

[0001272] • interface characteristics detection

[0001273] • interface characteristics selection

[0001274] • connection establishment

[0001275] • connection shutdown

[0001276] • send

[0001277] • receive

[0001278] For MPI, for example, some sample functions in these categories are:

[0001279] • MPI_init

[0001280] • MPI_Finalize

[0001281] • MPI_Get_version

[0001282] • MPI_COMM_size & MPI_Comm_rank

[0001283] • MPI_Comm_connect

[0001284] • MPI_Comm_disconnect

[0001285] • MPI_Send & MPI_Bsend

[0001286] • MPI_Recv

[0001287] Using sockets, these types of functions are performed by the following calls:

[0001288] • getsockname

[0001289] • getsockopt

[0001290] • connect

[0001291] • shutdown

[0001292] • send

[0001293] • recv

[0001294] For Tuxedo, the following calls are used to accomplish similar tasks:

[0001295] • tpinit

[0001296] • tpterm

[0001297] • tpbegin

[0001298] • tpcommit

[0001299] • tpcall & tpcalla

[0001300] Although for Tuxedo, the tpbegin & tpcommit calls also include the notion of a transaction, so that multiple messages (to multiple partner components, if desired) may be incorporated into a single 'unit of work', with the usual 'atomicity' guarantees.

[0001301] A common set of functions should be established, one for each of these categories of communications service, with the argument syntax sufficient to make later migration to either

MPI or Tuxedo a little easier. A complete 'metaAPI' may not be necessary, nor worth the effort. But some consideration should be given now to defining a set of standard 'system sockets lib' functions (methods) rather than have each subsystem implementing things in varying ways.

[0001302] These half-dozen (or dozen) routines can be built just twice, once for Java and once for C/C++, fairly quickly, then used as needed throughout the system. After all, it is not the functionality, but only a thin interface 'wrapper' function that will be required in each case.

[0001303] For example, to take the MPI_Send function and map it to the socket send call:

[0001304] MPI_Send(void *buf, int count, MPI_Datatype dtype, int dest, int tag, MPI_Comm comm) ;

[0001305] size_t send(int s, const void *msg, size_t len, int flags);

[0001306] would appear difficult. However, a simple function layer may defined between, such as:

[0001307] size_t eFsend(int s, const void *msg, size_t len);

[0001308] the use of which would make it possible to ease future migration efforts as well as simplify the interface for all subsystem authors. Note that certain of the arguments in the MPI call may be set internally by the interface layer and therefore not presented as 'public' arguments to the eFSend function. The MPI_Datatype dtype can be handled in this way: if all messages are transmitted as 'flat' sequences of chars, for example, this could be set to MPI_CHARACTER inside the eFSend function. The MPI_Comm comm 'communicator' indicator as well may be set internally to MPI_COMM_WORLD (the MPI default). Similar handling should work for the MPI args int dest and int tag, leaving void *buf and int count which map to the socket send function's const void *msg and size_t len, etc.

[0001309] This technique may require that the arguments not present in the eFsend wrapper call (the one to actually be used by client subsystems) will have to be handled (set and passed internally) within the eFsend function itself, possibly in cooperation with eFcomm library initialization. Finer grain-controls such as MPI_Datatype dtype may be needed at the individual subsystem level rather than being set internally by the eFsend function library routine on behalf of, and transparently for, each client subsystem. If such level of control is not needed, this approach should work.

[0001310] This will allow rapid implementation of a very thin layer that 'hides' the details of sockets functionality from all 'client' subsystems, so that future changes in communications routines may be mostly confined to the set of eFcomm API routines. For further 'protection' an 'escape' argument such as struct eFComm comm could be defined as in:

[0001311] size_t eFsend(int s, const void *msg, size_t len, struct eFComm comm);

[0001312] which will allow passing (in future) any arbitrary (currently unforeseen) collections of arguments required by future APIs. Given the foregoing discussion, the following baseline eFcomm library functions are proposed:

[0001313] For all functions, the structure 'eFcomm' is defined as:

[0001314] struct eFCommStruct

[0001315] {

[0001316] char[2] version ;

[0001317] char *placeholder

[0001318] } ;

[0001319] For all functions, all arguments except for eFcomm and any others noted have the meanings defined for the sockets library.

[0001320] A Java class or classes with corresponding functionality will be defined. It may be based, if possible, on the WebLogic 'rich' sockets interface.

[0001321] **eFcomm library functions:**

[0001322] _____

[0001323] function name	corresponding sockets	description lib function
[0001324]		
[0001325] _____	_____	_____
[0001326] 1. eFCommInitn/a		**- initialize the eFCommLib**
[0001327] 2. eFOpenConn:	**socket (n)**	**- Open a TCP network connection**
[0001328] int	eFOpenConn	(int domain, int type, int protocol, struct eFcomm) ;
[0001329]		where:
[0001330]		int domain == PF_INET
[0001331]		int type == SOCK_STREAM
[0001332]		int protocol ==
[0001333] 3. eFAccept	**accept (3xn)**	**- accept a connection on a socket**

[0001334] int eFAccept(int s, struct sockaddr *addr, socklen_t *addrlen, struct eFcomm);

[0001335] **4. eFBind** **bind (3n)** **- bind a name to a socket**

[0001336] int eFBind(int s, const struct sockaddr *name, socklen_t *namelen, struct eFcomm);

[0001337] **5. eFConnect** **connect (3n)** **- initiate a connection on a socket**

[0001338] int eFConnect(int s, const struct sockaddr *name, struct_t namelen, struct eFcomm);

[0001339] **6. eFGetConnOpt** **getsockopt (3n)** **- get options on sockets**

[0001340] int eFGetConnOpt(int s, int level, int optname, void *optval, socklen_t *optlen, struct eFcomm);

[0001341] **7. eFSetConnOpt** **setsockopt (3n)** **- set options on sockets**

[0001342] int eFSetConnOpt (int s, int level, int optname, const void *optval, socklen_t optlen, struct eFcomm);

[0001343] **8. eFListen** **listen (3n)** **- listen for connections on a socket**

[0001344] int eFlisten(int s, int backlog, struct eFcomm);

[0001345] **9. eFRecv recv (3n) - receive a message from a socket**

[0001346] ssize_t eFrecv(int s, void *buf, size_t len, int flags, struct eFcomm);

[0001347] **10. eFSend send (3n) - send a message from a socket**

[0001348] ssize_t eFsend(int s, const void *msg, size_t len, int flags, struct eFcomm);

[0001349] **11. eFCloseConn shutdown (3xn) - shut down socket operations**

[0001350] int eFCloseConn(int s, int how, struct eFcomm);

[0001351] **12. eFCommTerm n/a - terminate & release comm lib**

[0001352] **Operational Concepts**

[0001353] **DECREASE MEMORY REQUIREMENTS FOR THE PROCESS SCHEDULER**

[0001354] **Decrease disk I/O**

[0001355] In the original design, messages are assembled immediately by the Assembly Engine and forwarded directly to the Delivery Agent. If the Delivery Agent can not deliver the message, then the assembled message is passed on to the Delivery Scheduler to attempt a second or third delivery option. It is not clear how the Assembly Engine knows that a message was not successfully delivered and therefore needs to be passed to the Delivery Scheduler. Furthermore, assuming the that Assembly Engine decides to pass on ten percent of its messages to the Delivery Scheduler, then the Delivery Scheduler should either hold all those assembled messages in the memory or store them on disk. If you assume that a typical submission contains one million messages, and a typical assembled message is about 15 Kbytes, then there would be 1.5 GBytes of data per submission that would be passed on to the Delivery Scheduler. This is a large amount of data to store in memory-- or on disk. This can be avoided by designing the Delivery Scheduler such that it only holds the message templates, the recipient info, and the status of each message. This design would only require a Delivery Scheduler to hold something on the order of several megabytes of data per submission.

[0001356] **DECREASE LAN TRAFFIC**

[0001357] Decreases in LAN traffic can be realized by pushing the assembly process further back in the process. This means that the assembled messages pass over the network less frequently. For e-mail messages, the assembly engine may access the SAN and place the messages directly into the Q-mail queues, completely bypassing the normal network traffic associated with SMTP.

[0001358] **DECREASE WAN TRAFFIC**

[0001359] Reduced WAN traffic is desirable where the system of the invention is implemented in a highly distributed manner. The amount of WAN traffic may be minimized by never passing assembled messages over the WAN. This also minimizes the amount of database traffic by passing all needed data from the Pre-processor Scheduler directly to the remote Delivery Scheduler so that the Delivery Scheduler does not need to get that data from the database.

[0001360] **DECREASE DATABASE ACCESS, ESPECIALLY TABLE SCANS**

[0001361] **CREATE A WAY TO HANDLE ROLLOVERS**

[0001362] A watchdog process may be provided to perform a periodic table scan on the database to determine the state of the messages for a given submission and then initiate the second or third delivery option for a particular recipient. However, table scans are to be avoided as much as is humanly possible, especially on large tables. This may be avoided by assigning the task of tracking the messages for any given submission completely to the Delivery Schedulers. Since there is now one process responsible for tracking the delivery status of all messages, it becomes very clear how to handle rollovers. Any process in the system that

identifies the change of a messages state (OOB handler, Web Agents, etc.) can contact the Delivery Scheduler directly to communicate the messages status.

[0001363] **Network Infrastructure**

[0001364] Baseline message volume may require outbound Internet connectivity of T3 (45Mbits per second) or better. The Network is preferably redundant so that the failure of any one component (switch, router, firewall, circuit, etc.) will not result in a failure of the overall system. The network is preferably monitored for usage of each network segment on a daily, weekly, and monthly basis. E-mail can be generated and sent to the network administrators when the weekly and monthly performance graphs are created. The system preferably has the ability to monitor specific switch and router ports so that specific equipment network bandwidth utilization can be tracked/monitored. A separate VLAN may be provided for the management interfaces for all devices to be monitored.

[0001365] The system in its preferred embodiment is able to deliver customized messages at high sustained rates, e.g., rates exceeding 1 million 15KB message per hour. The architecture is distributed and modular in design and volume can be increased by adding resources (network bandwidth, processing systems, database machines, etc.). The system is preferably able to handle peaks of up to 4 times the average (or steady state) level of user traffic.

[0001366] Use of the system of the invention may be provided as a service to customer companies that desire to send large volumes of custom messages to their users but do not have the capacity to track and administer the actual delivery of the messages themselves. Such customers may include companies that have information which must be delivered via large volumes of time-sensitive messages. Potential users of the system may be, e.g., stock trading

companies, catalog companies, electronic bill presentment enterprises, retail outlets, financial institutions, publishing and global Fortune 1000 companies.

[0001367] While the invention has been particularly shown and described with reference to preferred embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made therein without departing from the spirit and scope of the invention.